Cooperation in Parallel

A Tool for Supporting Collaborative Writing in Diverged Documents

by

Benjamin Mako Hill

B.A., Hampshire College (2003)

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Masters of Science in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2007

Author____
Program in Media Arts and Sciences
August 10, 2007

Certified by____
Walter Bender
Senior Research Scientist
Program in Media Arts and Sciences
Thesis Supervisor

Accepted by____
Deb Roy
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Cooperation in Parallel

A Tool for Supporting Collaborative Writing in Diverged Documents

by

Benjamin Mako Hill

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on August 10, 2007, in partial fulfillment of the
requirements for the degree of
Masters of Science in Media Arts and Sciences

Abstract

This thesis builds on theory from the fields of computer supported cooperative work and computer supported collaborative writing and the example of work in the free and open source software development communities to describe "cooperation in parallel" — a mode of collaborative writing among parties with diverged goals and distinct documents. It examines the social and technological requirements of cooperation in parallel for writing and considers the effectiveness of existing technologies ability to address them. It then presents the design and evaluation of *TextNet*: a wiki for document writing that uses techniques from software development including branch creation and tracking features and introduces a text-specific interface for merging changes and representing and resolving conflicts. It concludes by arguing that TextNet makes important steps toward the support cooperation in parallel for writing.

Thesis Supervisor: Walter Bender
Title: Senior Research Scientist, Program in Media Arts and Sciences

Cooperation in Parallel

A Tool for Supporting Collaborative Writing in Diverged Documents

by

Benjamin Mako Hill

The following people served as readers for this thesis:

Thesis Reader____

Chris Csikszentmihályi
Associate Professor of Media Arts and Sciences
Program in Media Arts and Sciences

Thesis Reader____

E. Gabriella Coleman
Assistant Professor of Culture and Communication
New York University Department of Culture and Communication

Contents

List of Figures

Chapter 1

Introduction

New technologies for *ad hoc* and distributed collaboration have already transformed the way that software is produced. In *The Cathedral and the Bazaar*, Eric S. Raymond famously attributed the success of Free and Open Source software (FOSS) to intense, non-hierarchical, largely volunteer collaboration [58]. In his books on FOSS development methodologies, Karl Fogel attributes this successful collaboration, in part, to technical facilitation and, in particular, to source-control management (SCM) [26, 27].

Wikis have employed ideas and technologies originally employed for software development toward the successful support of collaborative writing on the web. In this thesis, I will explore the way that technology developed to support software development can be applied to the production of document-writing. In particular, I will describe the application of principles derived from programming communities who have created a new set of technologies called distributed source-control management (DSCM), designed for software development, to writing. I will argue that these tools facilitate an important new mode of collaboration. This new model of cooperative work, *cooperation in parallel* (CIP), describes joint work in groups working toward divergent ends. The result is that groups working toward separate goals can collaborate and contribute to each others projects in ways that strengthen and bolster their individual projects.

In the Chapter 2, I give three extended examples that demonstrate the utility of CIP. These diverse examples highlight the core issues, theoretical concerns, and design principles that inform this work and serve to ground the work in important real-world problems. In Chapter 3, I give a short overview of relevant theory and background from the fields of computer supported cooperative work (CSCW) and computer supported collaborative writing (CSCWr) and discuss SCM and DSCM technologies. I use the example of SCM and DSCM in software to show how DSCM is facilitating CIP and to suggest reasons why CIP is *not* currently practiced in writing communities. I offer a more reflective anthropological examination of the social differences between writing and writing code — a topic that underlies and effects much of my work on this project — in Chapter 4. In Chapter 5, I examine four widely used cooperative writing technologies — each representative of a class of tools — and explore the way that these technologies support and hinder CIP in writing. In Chapter 6, I describe the design and evaluation of TextNet, a system I have designed and implemented to support CIP for writing. Before concluding in Chapter 8, I present the results of evaluation of that system in Chapter 7.

Chapter 2

Extended Examples

Wikis provide a widely used platform for *ad hoc*, asynchronous, collaborative writing. As such, they provide both an obvious point of departure for the technical design of tools to support CIP in writing and an example of a tool whose communities may be well served by CIP. Tools supporting CIP for writing have applications in a variety of fields and communities. This chapter highlights three such communities currently struggling with traditional wikis in extended descriptions that serve to demonstrate both the scope and the details of problems that CIP addresses.

2.1 J8 Summit

The governments of Canada, France, Germany, Italy, Japan, Russia, the United Kingdom and the United States participate in the Group of Eight (G8) international forum which holds an annual ministerial meeting hosted by one of the participant countries and attended by heads of states representing each of the participant governments [50, 6]. The Junior 8 (J8) is designed to be a similar forum for children and young people of the G8 countries. In addition to hosting eight students between the ages of 13 and 17 from each G8 country, the conference is attended by ten additional delegates from the developing world. The most recent J8 was held between June 2 and June 9, 2007 in Wismar Germany [7, 71].

The goal of the J8 summit is the production of a series of "communiqués" by the student attendees on the subject of the global issues on the G8 agenda. Even in the process leading up to the conference, delegates are expected to write together as they research and prepare for the production of communiqués. The work at the most recent J8 was coordinated in a wiki — a modified version of the python-based *MoinMoin* hosted by UNICEF in New York City. Students needed access to the Internet while adding or modifying content to the wiki — a fact that limited involvement by delegates with less regular access to Internet connections.

Upon arriving at the J8, each student was given an early production version of the One Laptop per Child XO-1. However, while students were able to write on their laptops, the UNICEF J8 wiki remained the official site for collaborative work. As a result, students were only able to access the latest versions of documents while connected to the Internet. This proved problematic as the Internet connections at the conference were inconsistent and unreliable. While the conference was hosted in a library in Wismar with an ISDN line, students were only present at the library for 6 hours each day. Much of the rest of the time, the students were at their lodging venue — a steel-hulled boat moored in the Wismar harbor. Not only was there no Internet connection on the boat until one of the final days, the steel hull prevented the students' laptops from communicating directly with each other.

The mode and organization of collaborative writing at the J8 was largely *ad hoc*. Students picked or were assigned issues they felt passionate or knowledgeable about; most focused their energy on a small number of issues, and made smaller contributions on other issues. Students worked in small stable groups on a section or part of a larger document. For the most part, students were not assigned roles and were not given rules on what or where they should contribute.

Because so much of the conference happened while disconnected from the Internet — and by extension, from the UNICEF J8 wiki in New York — students wanting to work on something offline would have to create an "offline" copy. By doing so, they would run a high risk of creating conflicting changes either with other students working offline or with one of the "gardeners" of the UNICEF wiki based in New York, who would organize, copy-edit, and synthesize work on wiki during the delegates long "offline" periods.

While the MoinMoin wiki was able to help the delegates complete their tasks, it was poorly suited to the model of collaborative writing practiced at the J8. Most importantly, MoinMoin's centralized model made contributions difficult as collaborators had infrequent access to the Internet. Problems related to the need for connectedness were aggravated in the face-to-face meeting when the available time for writing far exceeded the time connected to the Internet.

The result was either inactivity or offline work that led to the possibility of divergence and, as a result, conflicting changes. There was no easy way to reconcile — or even recognize — these conflicts. The MoinMoin wiki has no method to merge changes or model them in a way that takes into account the non-linear nature of development. The lack of methods to help identify and merge changes led to a lack of offline work. Ultimately, the cost in time and energy of merging changes outweighed the benefit of working offline.

Similar problems exist in other wikis and in most other collaborative writing tools. Alternative real-time synchronous communication tools, where collaborators can monitor each other's work in real time (e.g., *Gobby* or *Google Docs*), would have been even more problematic in the J8 use-case, both in their increased reliance on synchronous connection and in their inadequate support for divergence.

An ideal collaborative technology for the J8 would be more friendly to offline and disconnected work, would facilitate tracking and merging divergent parallel versions of the document, and would allow for more efficient and more productive collaboration. To do so, it might borrow their wiki's low barrier-to-entry, web-centric *ad hoc* approach to work flow, and history-tracking functionality and would build on these features to allow cooperation in parallel.

2.2 Wikipedia Stable Versions

Many people, both inside and outside of Wikipedia, have expressed concern with quality assurance in Wikipedia articles. To address this issue, several dozen "article validation proposals" have been suggested in the Wikipedia community [5]. "Stable versions" is the term for one of the most popular models that is currently being implemented on the German Wikipedia with several other Wikipedia language groups watching closely. Stable versions aim to allow Wikipedia editors to vet and approve versions of articles that are then stabilized or frozen so that only site Administrators can modify them. After stabilization, work continues in a copy of the article earmarked for "development." In the model, changes are then "synchronized" from the development version to the stable version every several months. Any emergency changes to a stabilized article are to be discussed first and then integrated, by hand, by a site Administrator.

The introduction of stable versions has been controversial in Wikipedia. The most recent proposal was rejected in the English Wikipedia but accepted in the German Wikipedia (the second largest Wikipedia community). While early proposals advocated a purely social system based on existing locking mechanisms and predictable naming of articles, a new version of the Mediawiki software with rudimentary built-in support for stable versions is in development by the Wikimedia Foundation [77].

While proposals for stable versions of Wikipedia article differ, they each aim to provide a simple means to create diverged or forked articles; no proposal equates stable versions *only* with "locking" or "protecting" an article. In the stable version model, work continues after articles have been stabilized — it just continues in a separate version of the article. Similarly, no proposal treats divergence as permanent. In the idealized setting, development versions are improved, periodically reviewed, deemed stable, and used to replace the old stable version. At that point, a new development version is created.

Even proponents admit that this model is idealistic. First, no degree of vetting or review will leave a stable article immune to changes. As a result, fixes and small changes continue in the "stable" branch in parallel to work in the development version. As each branch develops, the result is two-way divergence. Since the sharing of changes will include both fixes to the stable version applied to the development branch, and uncontroversial changes to the development branch being applied to the stable version, simple difference representation to the branches under extensive work quickly becomes insufficient.

As the development version and the stable version diverge, the cost of merging changes between the branches becomes increasingly high since each change must be discussed and made "by hand" by administrators. Additionally, several editors have asked about what happens when the development version clearly deteriorates in quality. The obvious answer, to many Wikipedia editors, is to merge positive contributions in the development branch into the high quality, but out-of-date, stable version. However, the cost of finding the differences between the documents and merging them by hand introduces a significant and often prohibitively high cost.

The stable version proposals are, at their essence, a call for parallel versions of articles within Wikipedia. There is not necessarily a disagreement between editors working on stable and development versions — the editors will frequently be the same people. While obeying the criteria for inclusion of text in a stable article (e.g., uncontroversial corrections of mistakes, addition of important new information, procedure for major updates of a stable version), the articles are meant to be developed in parallel.

Calls for stable versions have been stymied, in part, by the raised cost in labor of synchronization of the the two versions or branches of the stabilized articles and the lack of a tool to present differences between diverged branches effectively. One editor objected to the proposal on grounds that it introduces, "more cumbersome stuff to juggle around manually." Many others echoed similar sentiments. The lack of a tool to track the relationships between the branches and assist in the efficient transfer of information between the branched versions creates an important barrier to a feature with widely recognized value in the community. An ideal tool would provide a means of representing differences between the two versions and allowing editors and administrators to easily apply and share changes between articles.

2.3 Wikinews

Wikinews, a wiki-based news website, facilitates collaboratively written news stories. The site was started in 2004 and is supported by the Wikimedia Foundation, the organization that also supports Wikipedia. Wikinews exists in 22 languages with a combined total of more than 37,000 articles — roughly one quarter of those in the English language version of the site. Unlike other Wikimedia wikis, Wikinews foundered early; producing an average of 16 articles each day in 2004, the site's output has stabilized at 4-6 news articles daily in 2007.

Axel Bruns has compared Wikinews to other collaborative "participatory" news sites like *Ohmynews* and *Indymedia*. He describes these other sites as offering, "multiperspectival coverage of the news.” Bruns concludes that Wikinews’ problem stems at least in part from its strong adherence to a principle of neutral point of view (NPOV) adopted from the Wikipedia project [16]: a complex concept that aims to ensure that a variety of different perspective are fairly integrated into an article in an unbiased way [32, 59]. NPOV requirements and guidelines in Wikimedia projects like Wikinews and Wikipedia serve to frame discussion so that users can work toward the goal of a single article that incorporates each of their divergent perspectives — a technical requirement for collaboration in wiki. Divergence is frowned upon and happens in separate articles or separate wikis.

Many commentators blame the consensus requirement for Wikinews’ difficulties. The Washington Post attributed Wikinews’ failure in part to the cumbersome peer-review process aimed to ensure NPOV [79]. Blogger Matthew Yeoman claimed that, “in their zeal for a Neutral Point of View, Wikinews seem to revise the life out of their stories, reducing lively news coverage to dull regurgitation of facts” [84]. Alex Bruns commented that, “while the Wikipedia’s NPOV doctrine is commendable in an encyclopedic context, then, it cannot apply in the same way for Wikinews; indeed, it may even be counterproductive where it stifles engaged debate on complex issues” [16]. In volunteer news production, more so than encyclopedia article writing, editorializing seems to play a more central, motivating role.

A real example from Wikinews can illustrate the way in which Wikinews is particularly poorly served by NPOV and the requirement that groups come to consensus on a single article. Adam, a new editor on Wikinews, wrote a draft of an article for Wikinews about how CNN had hired three right-wing commentators and about speculation in the press that this was designed to help the news network compete with Fox News, which had recently passed CNN in the cable news ratings in part due to its reputation for right-wing punditry.

Adam’s article was quickly and strongly attacked by several other editors on the site. Worried with the political subject of the article, one editor said, “this is not really news in its current form … because I still think its your opinion.” Editors suggested changes to the title, introduction, and other sections of the article which Adam disagreed with but ultimately incorporated in order to allow publication on the site. In his creation of two other articles — also on political issues — Adam ran into similar opposition from other editors. He became disenchanted with Wikinews and ceased contributing.

Both Adam's motivation and those of the other editors were politically colored. Adam self-identifies as a leftist and wanted to highlight what he saw as a dangerous shift to the right by CNN. Several of his article's most vocal opponents on Wikinews have strongly stated right-wing positions on their personal websites. Wikinews' adherence to NPOV is problematic in general because news — and political news in particular — is, unsurprisingly enough, frequently politicized.

There are, as Bruns describes it, two major models for collaborative and independent news production. Collaborative work toward the production of the single articles, like in Wikinews, or independent and fully separate production of several articles with the ability to respond and discuss [16]. Collaboration in parallel describes a third option well suited to a project like Wikinews.

In CIP, collaborators like Adam would write their article as they had before. If other editors disagreed or wanted to make substantive changes that Adam did not want to incorporate, they would have the option of making their changes in a separate branch of the article. As the articles develop and solicit feedback and copy-editing from the community, the two groups would be able to share those changes that apply to both versions and would be able to diverge where politics and aesthetics allow it.

Google News currently aggregates news from more than 4,500 English-language news sources. Major news stories will frequently be the subject of hundreds or thousands of articles visible in Google News — many of them representing only slight modifications to stories written by several major news wires. Traditional news reporting agencies diverge on the text of wire articles because they want to tailor stories to the interests and politics of their audiences. The vast multiplicity in audiences in contemporary mainstream news publishing is testament to readers' strong desire for this type of customization by news sources. While most encyclopedia readers are happy with a single encyclopedia from a neutral point of view, it seems likely that news readers will be more reticent. Wikinews' insistence of single articles over multiperspective news is unable to cater to the desires of writers like Adam and is poorly suited to producing the type of news that many news readers want.

Using CIP, Wikinews could produce not one a version of an article but many — and could do collaboratively. Users could choose between articles based on established points of view and on other readers' experiences and feedback. The lack of tools and technology to facilitate the type of collaboration in parallel in wikis necessary to bring this about means that Adam, and others writers interested in collaborative and wiki-like news production, simply don't write news collaboratively at all. It means that readers are left with their products.

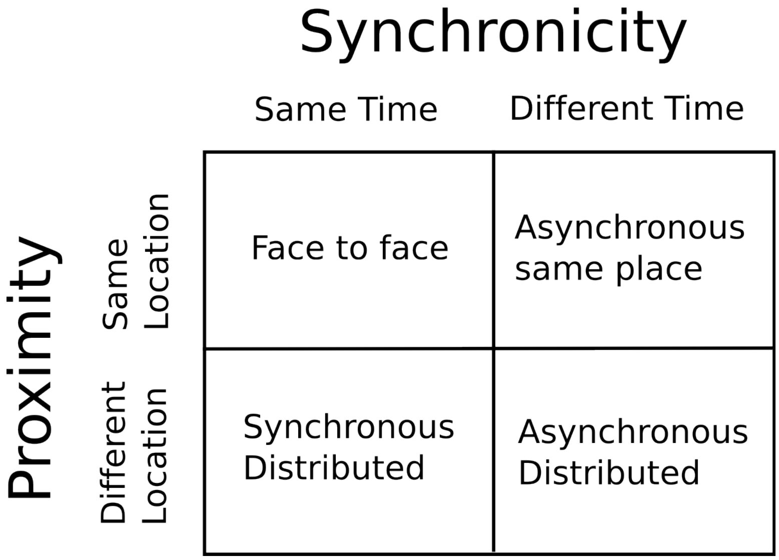
Chapter 3

Theory

3.1 Computer Supported Cooperative Work

Over the last two and a half decades, the focus of the research done in the academic field of computer supported cooperative work (CSCW), and in particular the sub-field of computer supported collaborative writing (CSCWr), have studied the technical support of collaborative textual production [65, 12]. An interdisciplinary research area involving the works of cognitive psychologists, organizational sociologists, and computer scientists, CSCW and CSCWr are methodologically most closely aligned with the human computer interaction sub-field of computer science, and most theorization is insantiated and tested in working system prototypes. CSCW theorists frequently categorize collaborative writing into four groups (see Figure 3-1) based on the modes of synchronicity and proximity of the work. By focusing instead on these theoretical goals, most collaborative writing systems can be placed into one of two alternative categories: those that seek to increase awareness and those that seek to support workflow [48, 66].

The first of these categories pertains to "awareness," an overwhelmingly broad term in CSCW that generally refers to, "an understanding of the activities of others, which provide a context for [ones] own activity" [22]. The term serves as, "a placeholder for [the] elusive practices of taking heed of what is going on" [63]. The theory behind awareness support is that if collaborators



Synchronicity

Proximity	Same Time	Different Time
Same Location	Face to face	Asynchronous same place
Different Location	Synchronous Distributed	Asynchronous Distributed

Figure 3-1: Breakdown of collaborative writing work modes (Adapted from Lowry et al. [48] and from Ellis, Gibbs, and Rein [24])

know what each other are doing, they are better able to coordinate and, as a result, can work together more effectively.

Classic examples of support for awareness in CSCW systems are Xerox PARC's *Cognoter* and *Media Spaces* projects which aimed to support collaboration by providing rich information (e.g., video, audio, text, graphics) to participants [68, 14]. Other efforts, especially those based on collaborative writing, have supported awareness by increasing the visibility of previous work. The *Read and Edit Wear* software — which showed areas of frequent editing through "wear marks" in a toolbar — and *Color Emacs* — which exposed document history through the use of color — are each classic examples of tools that support collaborative writing by exposing "history" of a collaborative document [36, 46].

A second body of work has aimed to support collaborative writing by providing support of time- and role-based workflow. In a time-based workflow system, a document or document part is "locked" while one collaborator edits it. Role-based workflow is focused on constraining collaborators' interactions by the nature of their contributions. Workflow systems are often modeled after collaboration in the traditional publishing industry: collaborators might be authors, editors, proofreaders, or others — all roles supported in different CSCWr theoretical frameworks and tools [48].[1] Together, workflow-based models play a central role in CSCW and CSCWr technology design [57].

The *Quilt* system from the University of Washington is an example of a role-based workflow system for CSCWr. In *Quilt*, users adopt roles which limit their ability to interact with the document (e.g., a reviewer might only be allowed to annotate the document). Another early CSCWr system, *MILO*, replaced *Quilt's* role-based system with one based more on a highly structured document and a flexible time-based model [38]. In both systems, collaboration is supported by using workflow to make the nature and timing of changes more predictable so that users can plan their work and interactions with collaborators.[2]

It is important to note that, in the technological support of both awareness and workflow, CSCW and CSCWr systems are designed to support the production of a single product in the context of shared goals. While these systems differ widely, this assumed goal is consistent throughout. Dillenbourg and Baker go so far as to define collaboration itself as, "a coordinated, synchronous activity that is the result of a continued attempt to construct and maintain a *shared conception* of a problem" [19]. Similarly, Lowry et al. define collaborative writing as, "an iterative and social process that involves a team focused on a *common objective* that negotiates, coordinates, and communicates during the creation of *a common document*" (emphasis added) [48]. The focus on supporting singular goals and singular documents has provided a point of unity in CSCWr. However, it has made the field slow to recognize an important model of collaborative work where collaboration may result in multiple non-canonical documents or documents that support multiple objectives.[3]

[1] Lowry et al. give an especially in-depth analysis and categorization of different types of role based workflow in their *Building a Taxonomy and Nomenclature of Collaborative Writing to Improve Interdisciplinary Research and Practice* [48].

[2] Lowry et al. also go into depth on the nature of planning and different methods of planning that different CSCWr methods attempt to support [48].

[3] My own work, including the work described in this thesis, has encountered resistance in the CSCW community due to the fact that it deviates from this core quality. One prominent researcher told me that, while potential useful, CIP was "not collaboration" and an inappropriate site for research in the community.

3.2 Revision Control for Software and Text

Since many of those building CSCW tools are programmers, it is unsurprising that a large portion of CSCW experiments and tools are designed to support the process of writing software. One key class of tools built to support collaborative work on software is source-control management (SCM).[4] SCM automates the process of storing and retrieving different versions of a piece of software — often using storage space- and transfer-efficient differential-storage systems. The first widely used SCM was the *Revision Control System* (RCS) written by Walter F. Tichy and published in 1982 [69]. Variants of Tichy's system are still widely used today and most modern SCM systems incorporate ideas from RCS. However, more important than RCS's technology was the fact that the system served to lower the barrier to collaborative software development. By being able to "revert" to any previous version of a piece of software, differences in opinion and conflicts among software developers were not reduced, but they became trivial to recover from. Particularly in the case of geographically distributed collaboration, RCS and its successors played an essential role by documenting software's "history" and providing a way for collaborators to monitor changes made by others.

SCM ultimately played an important technical role in facilitating the large-scale collaboration in FOSS and the creation of the GNU/Linux operating system, the Apache web server, and many other important software projects [26, 27]. RCS, itself free software, was incorporated into Dick Grune's *Concurrent Versions System* (CVS) which added many features and enjoyed almost universal use in FOSS communities. In particular, CVS added "lockless" operation to RCS: the ability to work on a file or project without setting it off limits for other collaborators. Lockless operation, in part facilitated by methods for handling merging and the resolution of inevitable conflicts, allowed for unprecedented asynchronous and entirely disconnected collaboration on software.

The effects of these features were profound. Collaborators were frequently able to work productively while disconnected, out of sync, and with little inter-collaborator communication. In

[4] SCM is frequently referred to using a variety of names. These include "source control," "revision control," and "version control." I use SCM in large part to emphasize the design of these tools for software source code and to distinguish them from tools which try to track or control revisions or versions of other types of creative works.

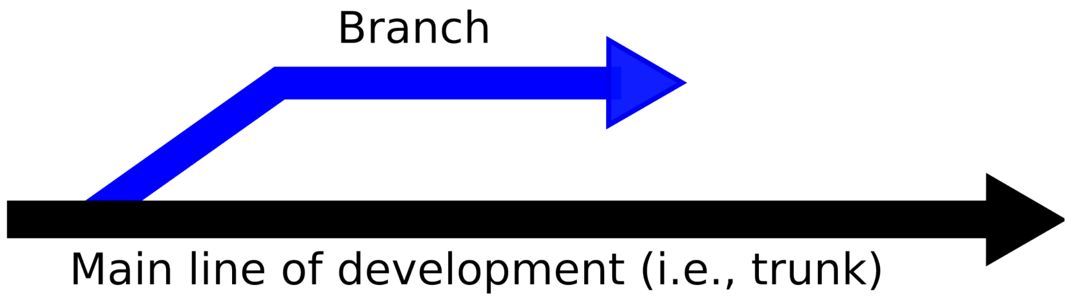


Figure 3-2: Visualization of a simple branch

this way, SCM helped pave the way for the success of FOSS projects, many of which are widely distributed, Internet-based, and built largely from volunteer contributions. Lockless SCM — and CVS in particular — has been cited by technologist David A. Wheeler as one of the most important software innovations of all time [82].

Noting superficial similarities between source code and documents — both are, from a purely technical perspective, text — ideas and technologies from SCM have been applied to the production of documentation and writing as early as Theodore Nelson's Xanadu [53, 52]. In fact, explicit attempts to introduce traditional SCM technologies to collaborative writing have been a major thread in both the CSCWr research and practitioner communities [20, 21, 40]. Wikis, perhaps the most recognized example, have incorporated SCM features as part of a sometimes explicit attempt to support FOSS-style collaboration in writing [17, 80, 41].

3.3 Distributed Source Control Management

In simple SCM environments, revisions are modeled as a serial list of changes. In reality, work is asynchronous, workflow is unenforced, and collaboration happens in parallel. To account for this, early SCM systems, including RCS, incorporated a rudimentary concept of "branching" [69]. Branching (or forking) refers to bifurcation in the history of a document often represented as shown in Figure 3-2. The term is designed to evoke a simple "trunk" and "branch" mode of work.

In early SCM systems, including RCS, branches were a numbering scheme that ensured "forked"

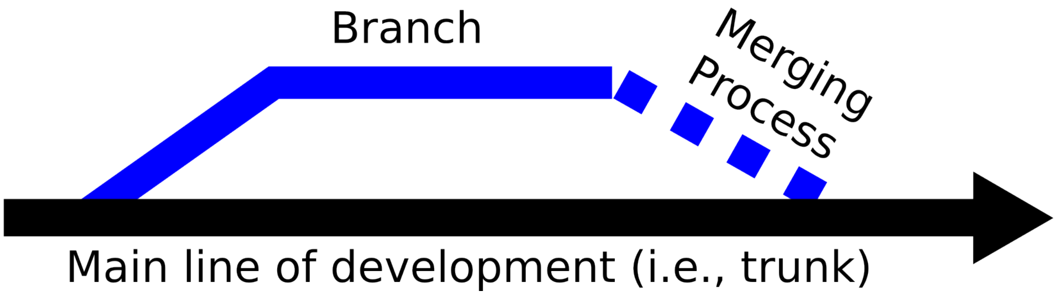


Figure 3-3: Visualization of simple branch and the merge process back into the trunk

versions of a file were marked so that users could determine when a branch had diverged. With time and with the help of rudimentary merge functionality in lockless systems like CVS, collaborative development began to make increasingly sophisticated use of branches to coordinate feature development, invasive or disruptive changes, and to coordinate releases [26, 27]. While the mode of production and the storage of revisions remained centralized, different branches existed within the centralized repository to allow disruptive work to move forward in parallel. When disruptive changes were finished, work was merged back into the trunk as shown in Figure 3-3. While CVS and other early SCM systems provided support for tracking branches, they rarely provided more than basic support for merging. The result was infrequent use of branching functionality and short-lived branches.

A new class of distributed source control management (DSCM) tools were created to facilitate more frequent and extensive use of branches. By increasing use of branches, the authors of DSCM tools aimed to support increasingly distributed, *ad hoc*, and disconnected collaborative software development. Invented only several years ago, these tools, and the ideas behind them, have already had rapid uptake in software development communities and in FOSS communities in particular. Examples of DSCM tools include BitKeeper, DARCS, Git, Bazaar, GNU Arch, Monotone, Codeville, Mercurial and SVK.

DSCM is distinguished from traditional SCM in several ways:

- There is no “trunk” or canonical version. Instead, every repository is a first-class branch and contains a full copy of relevant history;

- Because there is no central repository, new peers can freely participate. These peers do not need write permission to any existing repository; they can make changes and commits into their own branch;
- Consequently, merging plays a central role in the day-to-day operation of the system; ordinary patch exchanges and updates are merges;
- Due to the increase emphasis on merging and the difficulties of conflict resolution, tools put increased effort on conflict reduction. Frequently, this involves improved history accounting and modeling [10].[5]

While the technical advances behind many DSCM systems are impressive, their significance lies in the fact that they allow for new, powerful forms of collaboration. These fall into two major categories:

1. The ability to work disconnected from each other and to sync with each other, in whole or in part, in an arbitrary and ad-hoc fashion;
2. Support for differences or divergence between two branches that can be maintained over time [35];

The most famous user of DSCM (and the source of one of the most widely used tools) is the Linux kernel project. Linus Torvalds, the primary coordinator of development on the Linux kernel has authored *Git*, one of the most widely used DSCM tools, and has described DSCM as "fundamentally [making] us (and me in particular) change how we do things. That ranges from the fine-grained change set tracking to just how I ended up trusting sub-maintainers with much bigger things, and not having to work on a patch-by-patch basis any more" [70]. David A. Wheeler describes how:

> Decentralized development has its strengths, particularly in allowing different people to try different approaches (e.g., independent branches and forks) independently and

[5] A in-depth description of the process of conflict resolution is included in both Chapters 5 and 6.

> then bringing them together later. This ability to scale and support "survival of the fittest" is what makes decentralized development so important for Linux kernel maintenance [81].

FOSS projects have grown increasingly dependent on capabilities of DSCM to engage in long-term divergence of a type that was previously impossible. This type of *ad hoc* parallel divergence has, to date, no obvious corollaries in writing.

3.4 Cooperation in Parallel

A new generation of FOSS projects, including the Linux Kernel, have employed DSCM in a set of promising new software development methodologies. One of these models has played a central role in the development of the Ubuntu GNU/Linux distribution. By diverging from the Debian project, an extremely large and complex code base, Ubuntu has been able to quickly create a successful operating system with a large install-base with a team of less than two dozen programmers. Using DSCM through a process described in Section 5.1, Ubuntu developers continue to share code with the Debian project in their parallel branches over time through published list of changes to packages and patches in Debian's bug tracking system [35]. While diverging or "forking," is frequently easy, *ad hoc*, bi-directional collaboration in areas of agreement within these diverged projects has historically proved difficult and labor intensive. Ubuntu's use of DSCM is noteworthy because it allowed a small team of developers to accomplish what, in most examples in the past, would have introduced a prohibitively high cost in time and effort.

Parallel development, as practiced by a series of GNU/Linux distributions including Ubuntu, constitutes an important new development model and a new mode of computer-supported cooperative work. This model, which I call *cooperation in parallel* (CIP) refers to *ad hoc* cooperative work among parties working toward different but partially overlapping goals. Navigating a constant tension between collaboration on overlapping goals and the need to protect and preserve divergence, CIP is grounded on collaborators' attempts to maximize both cooperation in areas of overlap and divergence where and when it is useful. When supported appropriately, CIP benefits all participants precisely because it allows for differences in their individual needs andlabor while summing those that are shared.

CIP differs from other forms of cooperation in that it does *not* rely on workflow or a shared conception of a single end-goal. Instead, CIP is entirely *ad hoc* and is always in the interest of the *individual* collaborators working toward their *individual* goals in an ecosystem of others pursuing related and overlapping goals. I hypothesize that the example of free software communities — and DSCM in particular — demonstrates that resonant divergence in CIP can be facilitated through technologies built around five design principles:

1. Supporting parallel work among individuals or groups interspersed by merge sessions. During merges, users are presented with work from collaborators and are asked to evaluate and integrate relevant work;

2. Emphasizing transparency over explicit forms of communication, modularity and traditional forms of awareness [11];

3. Effectively displaying and summarizing changes between documents in ways that take into account previous decisions to diverge;

4. Representing, reflecting, and resolving conflicts upon merges;

5. Allowing collaborators to quickly and easily decide when and who to work with. Toward this end, collaborators should easily be able to determine who appropriate collaborators are;

The first criteria — support for parallel work interspersed by merge sessions — describes the overall model for CIP. It is the relationships of work in branches, but with points of at least partial convergence, that forms the most superficial description of CIP. However, this criteria alone might describe much traditional work in branches as well. The second criteria — emphasis of transparency — ensures the *ad hoc* nature of collaboration which serves to distinguish CIP from other types of cooperation through branches. Cooperative work in branches can be supported through workflow, modularity, communication, and awareness — as is the case in many of the

CSCW projects described in Section 3.1. Transparency, in this case, means collaborators are given free reign to diverge without consideration for or communication with other collaborators. While they are able to see, borrow, or share at will, they are never forced to work in a particular way or be forced to track the contributions of a potential collaborator.

The third and fourth criteria — effective display of changes and support for the representation and resolution of conflicts — are necessitated by the first and second criteria taken together. If work is going to be done in branches and if that work will be largely uncoordinated, the branches will diverge. Any effective CIP system, then, must put a strong emphasis on communicating the nature of this divergence to users when and where desired. Similarly, while many other CIP systems use workflow and communication to avoid conflicts in the first place, CIP's *ad hoc* nature ensures that they are always possible and may, in some cases, be extensive. As a result, CIP tools must be able to effectively and appropriately represent conflicts to users and provide methods to resolve them.

The nature of the cooperation implied in the first four criteria aims to reduce the cost in labor and cognitive load of *ad hoc* divergence of branches but it does not eliminate costs altogether. As branches diverge, the cost of collaboration may become prohibitively high for some collaborators in some contexts. Similarly, in cases where previously unrelated branches converge, new collaborative relationships may be initiated. The fifth criteria states that this process, in addition to the sharing of changes described in criteria two, should be given special attention in CIP.

There are several tools used to support CIP in software source code. Each of these criteria is satisfied by each of these tools being employed by communities working effectively in parallel. Taken together, these criteria can effective divide those working on traditional branches built around canonical "trunks" from those who are truly working in parallel. As a result, I believe that tools trying to support CIP in other areas can be evaluated in their ability to satisfy these principles.

3.5 CIP for Writing

While several tools and algorithms exist for computing, minimizing, and representing differences and conflicts, there have been few attempts to do so in ways that are designed for writers. While several tools exist for representing and resolving three-way conflicts, they are developed for and integrated into programming environments. These tools represent changes on a line-by-line basis which, while appropriate to source code, is ill-suited for writing where the most useful units are words, sentences, paragraphs and sections. More importantly, they rely on notation and interfaces that are prohibitively complex for non-programmers. Even using these tools, conflict resolution in three-way merge environments is famously difficult for technical users.

CIP has been largely limited to software development communities. To date, there have been no attempts to support "parallel development" for writing or literary works or to apply the tools and principles of DSCM toward the facilitation of collaborative writing. A writing-specific interface to the branch-management functionality of DSCM, with special attention paid to conflict resolution in an asynchronous, distributed, environment, will allow collaborative writing in ways and among groups where it was previously impossible.

When used non-collaboratively, such a system might make the maintenance of partially diverging and partially overlapping copies of a resume, curriculum vitæ, or grant proposal easier. In the context of collaborative projects, like wikis, it will decrease the cost, in time and effort, of staying in sync with diverging versions of articles.

It might, for example, provide an alternative to the often-contentious "Neutral Point of View" in Wikipedia and Wikinews [75, 59] and allow for collaboration among individuals or groups with *different* points of view in areas of potential overlap (e.g., in Larry Sangers recently announced *Citizendium* project[6]). In each of these cases, CIP will allow writers to diverge where necessary while collaborating where possible. I believe that this has not been attempted due in part to a lack of methods and tools that allow users to stay up-to-date with what other diverged collaborators have done provide text-appropriate methods for conflict resolution.

[6] More information on *Citizendium* is available on the project website at: `http://citizendium.org/` and `http://citizendium.org/essay.html`.

Chapter 4

Writing and Writing Code

Source code is superficially very similar to literature. Both are constituted by letters, numbers, and punctuation. Both are written in "languages" and by "authors." These technical similarities supports an analogy that in turn supports the design of TextNet: software to support a mode of writing inspired by software development methods and using similar tools.

When pursued, the analogy is deeper, more nuanced, and less straight-forward: both software and documents are expressions of ideas and reflections of their authors' complex social relationships — relationships that differ between and within each group. For example, CIP is possible in FOSS communities in part because there are a set of community norms that facilitate and encourage this type of collaboration. FOSS's large community and strong voluntary nature have led to a set of design principles for FOSS technology like SCM and DSCM. FOSS is fertile group for CIP because its communities play out a tenuous balance between the agreement necessary for broad collaboration and the constant possibility of divergence.

In this chapter, I compare code and text while focusing on the communities that create them and the way that these differences frame and inform collaborative work. I attempt to harness an understanding of the social arrangements around CIP for code to draw analogies to CIP for writing and to establish a set of non-technical criteria for its support.

The fundamental difference between writing code and writing text, I will show, is one of community — of producers and of audiences. Supporting CIP for writing is about supporting the communities and processes for the production of documents and about understanding the communities of collaborators and readers that motivate and support the creation of works. By problematizing our conceptions of programming practice, by learning from the examples of programmers, by analogizing programmers and authors, and by understanding both why and how texts are produced collaboratively, we can begin to imagine one more component of what CIP might look like.

4.1 Coded Communication

There is a pervasive argument, primarily held by non-programmers, that software is an entirely functional, non-creative work. Instead, the argument goes, it is merely a tool whose design reflects and is evaluated in its ability to fulfill its function — comparable to the design of a circuit or a hammer. While widespread, this position presents both a false dichotomy of function and expression and an incomplete account of the expressiveness of code. It is held by very few experienced programmers.

In a series of court cases, lawmakers and the government defended these positions against a counter-discourse by programmer-activists attempting to establish source code as a legal — and thus constitutionally protected — form of expression. In the landmark *Bernstein v. US Department of State* case, computer programmer Daniel J. Bernstein sued the government over laws barring export of cryptographic source code on the basis that they constituted a prior restraint on free expression [1, 2, 3]. In arguing against this position, the U.S. government offered a strong version of the argument that source code is fundamentally distinct from writing by claiming that, "a description of software in English informs the intellect but source code actually allows someone to encrypt data" [1]. The government argued that code was without aesthetic or communicative properties and compared is to a circuit diagram, an equation, a bridge, and a hammer as a way of highlighting the way that these are distinct from protected forms of expression like many forms writing.

Ultimately, the government's arguments failed to convince the court. In its decision, the judges sided with Bernstein offering an eloquent description of source code comparing it to art and to writing:

> The music inscribed in code on the roll of a player piano is no less protected for being wholly functional. Like source code converted to object code, it "communicates" to and directs the instrument itself, rather than the musician, to produce the music. That does not mean it is not speech. Like music and mathematical equations, computer language is just that, language, and it communicates information either to a computer or to those who can read it [1].

With its analogy to piano rolls, the court explained that all communication is, to some degree, coded and that many types of speech — and writing in particular — is functional in nature. In the view of the court, being largely functional did not necessary distinguish code from other types of creative works.

Source code, like law or musical annotation, is a particular, formal description that makes saying certain types of things to certain groups of people easier. With its emphasis on procedural integrity and its attempt to reduce the possibility of ambiguity, it is much easier to build machine code from C source than from unconstrained English prose — although programmers try to do the latter as well. Comparisons between different formalities — be it a comparison between US Legal English and Java or Java and Python — result in the description of formalities that make it easier to say certain things, to certain people, and to endow language with certain effects.

4.2 Code and Aesthetics

In Bernstein, the Ninth Circuit court found that software was protected because it was as communicative as other forms of protected expression and that it was not fundamentally different from other forms of writing and expression. In his PhD dissertation at the English Department at the University of Pennsylvania, Maurice Black examined and exposed the literary nature of source code and described much deeper similarities between the two modes of creative production by focusing on the role to which source code is an aesthetic and literary artifact [13].

Black's argument is most strongly supported by testimony of programmers through statements that equate programming with literary pursuits like poetry. Black's analysis spans the full history of computer science beginning with Lady Ada Lovelace and ending with the iconic computer scientist Donald Knuth. Black's examples are most explicit with Knuth who describes programming as, "as an aesthetic experience much like composing poetry or music, "and as, "the process of creating works of literature, which are meant to be read" [13]. Black demonstrates that Knuth's description is not mere analogy. He argues that Knuth's goal is to correct the type of mistake made by the government in Bernstein — focused on the way that software is run by computers, the government lost site of the fact that source code is written *by and for humans*. Black argues that through this process and through others like it, code becomes an essentially literary work. As Black shows through a treatment of John Lyons' famous pedagogical annotations of the UNIX source code, software offers the potential for close reading and as much aesthetic bias as in more recognizably literary texts.

In a forward to the computer science book $A = B$, Donald Knuth writes, "science is what we understand well enough to explain to a computer. Art is everything else we do" [56]. Proving this point, a quick survey reveals the vast multiplicity of ways to "explain" even a simple process to a computer. Wikipedia lists 535 "notable" programming languages and an additional 231 variants of BASIC [4, 8]. The esoteric programming language wiki lists another 360 "esoteric" languages — most of them designed for fun or to prove a point [25, 49]. A project trying to collect implementations of programs that can output the full lyrics to *99 Bottles of Beer* has collected implementations in more than 1,000 different languages [62]. The computer acts indistinguishably in each of these 1,000 examples — it is the experience for the human programmer or reader that differs and it is on the level of aesthetics that many of these languages, and of the programming communities and cultures that have created and sustained them, diverges.

In his article on *Software as Art*, Gregory Bond points out that that programmers evaluate not only source code but software products themselves using highly aesthetic criteria that include "correctness, maintainability, readability, lucidity, grace of interaction with users, and efficiency as properties that inspire pleasure, or pain, in [the software's] readers" [15]. David Hillel Gelernter describes the role of aesthetics in depth in his book *Machine Beauty: Elegance and the Heart* *of Computing* where he demonstrates how it was the, “lust for beauty and elegance [that] underpinned the most important discoveries in computational history” [29].

Just as aesthetic sense informs language, idiom and figures of speech for both readers and writers of poetry or novels, programmers part ways with each other and form communities around languages, coding standards, and programming style — often based on criteria that can only be described as aesthetic. These types of divisions lead to and differentiate separate programming communities — and literary communities as well.

4.3 The Particulars of Programming

In *Code and other Laws of Cyberspace*, Lawrence Lessig argues that computer code and laws — i.e., legal codes — are both best considered as sets of “proposed rules” [43]. Lessig’s book explores an extended analogy hinted at by the Ninth Circuit court in Bernstein: code is like other types of writing in that both are subsets of a class of creative, expressive works designed to concisely and unambiguously convey a set of rules in a specialized language written for a small community educated to understand them. Furthermore, “speech acts” play an important role in both law and code and, when supported with proper equipment — a government or a computer — can effect real change [64]. As a frequently cited area of convergence between code and writing, the practice of writing law provides a rich space for the examination of important distinguishing characteristics of writing and programming communities.

James Grimmelmann, a programmer and professor of law, has ruminated on the differences between law and code at several points in his career. While Grimmelmann agrees that law and code are deeply similar, he explains that while in software nuances are complex and counterintuitive, their encoding, in the context of a particular compiler or computer is fundamentally predictable. This differs from law where, he explains, “meanings are often negotiable and the most formally correct reasoning can be undone if the result would be unjust or inconsistent with a statutory scheme as a whole” [31].

Jeff Lipshaw, another law professor, references Wittgenstein to try and take Grimmelmann’s argument one step further by arguing that programming is missing perlocutionary acts, “family resemblances," inductive rule-following, and that while, "human beings who design the program may disagree as to what the program was intended to accomplish, the program itself can only serve one master" [47]. Lipshaw overstates his case about software's self-contained nature; in fact, programs frequently are designed to operate differently in different contexts. He also overstates the lack of opportunistic design in computer software; "polyglots" provide an important counter-example through their design to run, either identically or with differences, in different programming languages [9]. But, Lipshaw's core argument — echoing Grimmelmann's — exposes an important and real difference.

The source of the similarity between programming and law, when compared to other forms of writing, is at the core of the most important difference between the two. Computer programming, like formal logic, is designed to be internally consistent and testable by effectively identical and identically programmed computers throughout a programming community. The use of computerized compilers and interpreters "codifies" programming languages and other software design decisions in a way and to a degree that is unique to works that are written to be parsed, validated, or executed by a computer. Ultimately, law privileges inherently personal judgment calls about fairness and justice over deductive reason — a decision a compiler *itself* cannot make. The result is a degree of unstated and accepted ambiguity in most forms of writing that does not — and cannot — exist for programmers at the point of compilation. This fact, while not always as significant as the government argues in Bernstien, lies at the heart of communities' ability to collaborate and to diverge. It places a limit on at least one class of divergence in programming communities and in their products.

The borders between different programming languages — and even different versions of one programming language — are severe at the point when a program is run. Divergence, including the creation of programming language dialects, is still possible in computer languages and happens frequently. However, even minimal divergence on the language level can result in complete incompatibility at the point of compilation and, as a result, on the level of textual collaboration. Even in situations where communities continue to work together, the result of this type of divergence for computer languages is different texts and different programs.

The fact that certain types of differences result in barriers to collaboration shapes the nature and degree of divergence. The desire to sustain textual collaboration through the use of libraries and code sharing and through access to large programming communities frames collaborative work on software. The resulting model supports a type of common-ground that makes collaboration possible. Paradoxically, it also opens the door to collaboration among groups who have reason to diverge.

4.4 Divergence and Collaboration

An understanding of the social nature of communities provides insight into how communities are divided and on the terms that make cooperation possible. CIP is the description a community of communities made of distinct groups — a network of networks of collaborators. As established in Chapter 3, CIP requires:

- A divergent conception of the ends — users must disagree on the end product;[1]
- A common process of textual overlap and steps toward the divergent ends;
- An ability and desire to work as part of an ecosystem or group of groups.

An analysis of the way that these qualities frame the social nature of CIP in software helps describe qualities of the social arrangements that may be important for CIP writing.

4.4.1 Divergent Goals

While CIP has played an important role in FOSS communities, it is inappropriate for most software development projects. Fundamentally, CIP is only appropriate in situations where users have different goals. Software development communities have shown that CIP is less useful and less likely where users are in complete agreement about their immediate ends.

[1] In a sense, disagreement on ends may be more common in CSCW and CSCWr than the disciplines and their technologies might imply. However, because of the way that CSCW and CSCWr frames collaboration, groups are forced to compromise on shared ideals.

One frequent source of shared goals is the fact that many programmers are paid to write together. When a firm builds a piece of software, a executive or project manager dictates feature goals to developers. In these situations, there is no need to diverge because programmers are not empowered to choose direction for the project or to decide to work separately in ways that might contradict the vision of the manager and that might lead to duplication of work. Common in firm-managed software development but also common in FOSS communities, the use of style guidelines (e.g.,Mats Henricson's *C++ Rules and Recommendations* or the GNU style guidelines [33, 28]) provide a way of standardizing more aesthetic qualities of software development. In contrast, CIP has flourished in FOSS communities where a majority of work is voluntary [42].

Writers will face a similar situation in CIP. While collaborative writing is widespread — by some estimates up to 70% of writing is done collaboratively — the vast majority of this happens in firms [65]. CIP will be more successful in communities where participants are free, even encouraged, to pursue their *own* ends. This seems less likely in established collaborative writing genres like a corporate annual report. While hugely collaborative, legal requirements, efficiency, and tradition dictate that there can only be *one* version of these types of documents.

As in software, CIP in writing needs to be driven a desire or need to diverge. CIP may play an essential role for a series of related but individually targeted fund raising pitches or grant proposals in a non-profit organization or descriptions or manuals for a variety of related but slightly differing products. In each of these examples, there is a strong desire to produce related but separate, tailored documents that would encourage users to diverge.

4.4.2 Common Process

Of course, CIP must balance divergence in ends with a shared vision of means. For CIP in software, this shared set of steps must involve shared code; for writing this must involved shared text. As a result, CIP for writing is impossible in areas where the goal *and* the means have diverged significantly. In programming this blocks collaboration between work in different languages. While ideas may be shared in these examples, no code can be. Similarly, use of a particular set of libraries or frameworks within a program can provide barriers to collaboration with similar projects that made different choices.

In Chapter 3, I described the way that CIP has played an important role in the creation of new GNU/Linux distributions. One widely cited example of such a collaboration was between the Ubuntu and Linspire projects [74]. However, cooperation was only possible between these groups because each project is based on Debian. Ian Murdock's *Debian Core Consortium* (DCC) aimed to facilitate CIP between a host of different distribution but only because of the code the distributions held in common [73]. While other distributions based on Red Hat, Slackware, or SuSE might have liked to collaborate in the DCC, the lack of a shared set of steps toward their individually distinct goals — i.e., a shared basis in code — rendered cooperation in parallel impossible.

As I argued in *To Fork or Not To Fork*, CIP has played a larger role in divergence of GNU/Linux distribution in FOSS communities than in the production of other types of software [35]. This desire to diverge and to work in parallel, rather than to build from scratch, is due in part to the central role that operating systems play in computer software. It is also due to the cost of reproducing the work of distributions which frequently contain the work of many thousands of of collaborators [35, 30]. For most people who want to create a new distribution, the immensely and usually prohibitively high cost of creating a distribution from scratch makes divergence a more attractive option.

Similarly, CIP for literary works will require not only the presence of shared goals but a strong need for shared text in the process. In much fiction, for example, borrowing is common but primarily happens on the level of themes, plot devices, or ideas; CIP will be of little use in these cases. The process of updating or building reference works, position statements, textbooks, curricula, and documentation provides good examples of literary works where there is a strong advantage to authors who are able to build off of previous work and where reproducing previous work is discouragingly high.

4.4.3 Ability to Diverge

Finally, CIP requires that users have the *ability* to diverge. Barriers to divergence come in the form of technical barriers, legal barriers, and social barriers. While DSCM has successfully addressed some of the technical barriers to divergence for code, social and legal barriers remain. Many programmers seem to be naturally hesitant to collaborate or borrow even at the cost of high reproduction of labor — a situation sometimes referred to as "not invented here" (NIH) and jokingly described as a syndrome. NIH syndrome is documented in a variety of situations and describe a complex desire for recognition and control over software that results in unnecessary reimplementation of available functionality. However, NIH is merely one example of a series of social types of hostility toward knowledge sharing and collaboration within organizations that might otherwise be working together [37].

Legal barriers act as another major barrier to CIP. Before any form of cooperation can take place, divergers must have *legal* permission to modify or incorporate their collaborators' work. Direct collaboration often simplifies situations around permission-asking and copyright: parties collaborating traditionally make explicit agreements to work together and, through the use of contracts and copyright assignments, can resolve legal ambiguity about control over copyright ownership. This reasonably burdensome process may appear less valuable in CIP where collaboration, and its benefits, are less direct. CIP in many free software communities is only possible because of strictly enforced definitions of freedom at the heart of FOSS including the *Free Software Definition* and the *Open Source Definition* that ensure that everyone with a copy of the software has the ability to create derivative versions [67, 55]. These definitions require copyright licenses that explicitly permit unrestricted copying, modification, and collaboration on software. As patents have become an increasing threat to software, software licenses have been rewritten to provide methods for addressing these legal barriers to sharing and collaboration [67].

The social and legal barriers to divergence for writers of literature is more formidable than the barriers for programmers. Historically, free software was an attempt by Richard Stallman to reclaim or reinstitute a model of collaborative software production he had enjoyed in the 1960s and 1970s [83]. With hundreds of years of copyright law and widely held conceptions of authorship that privilege a Romantic conception of authorship [34], there are immense barriers to analogous types of literary collaboration. Divergence is simply not a choice for most writers.

CIP will be more likely in communities with established collaborative work where these social and legal barriers have been addressed or in new communities with different social conceptions of authorship and different copyright norms. The Definition of Free Cultural Works [51] and Creative Commons argue for licenses appropriate to literary production that explicitly allow for divergence. Wikipedia and other wiki-based projects offer examples of rapidly expanding communities with social norms that may be supportive of CIP and that have embraced a licensing model that removes legal barriers to divergence.

4.5 Synthesis

Not all code is written in parallel. Not all code is even written in processes that are particularly collaborative or conducive to cooperative work. The idea that all writing would benefit from CIP is not one that I advance here. While CIP has improved FOSS communities' ability to maintain divergence, its most important role has been in promoting new types of projects that were impossible without CIP. It seems reasonable to assume that CIP for non-software will follow a similar path. The question, therefore, should not be, "can CIP produce a great novel?" but rather, "what current genres could be enhanced by CIP?" and "what new literary genres will CIP produce?"

Writing and coding communities are too large and diverse to make meaningful broad statements about either or analogies between the entirety of both. One can, however, establish that writing and code are both defined — if not always defined similarly — through their constituent communities. It is these communities and their norms, laws, and technologies that provide impetus, facility, and constraints on cooperative work. It is through these particularly structured communities of practice in free software that CIP is facilitated. It will be through analogous structures that an analogous form of CIP for literature will be supported.

In a sense, writing provides space that is at least as fertile as programming for CIP. In that goals in writing are, if anything, less further aligned than in programming, the desire to diverge, for a variety of reasons, is a strong one. On the other hand, any collaborative writing process must address a long history of approaches to writing that are frequently at odds with cooperative work.

While CIP in code has not always resulted in a more efficient ways of building software, it has empowered people to create software of types, in ways, and in situations where it would not have been possible before. FOSS has not, for example, revolutionized the oldest and most established areas of software development — databases, compilers, and customer relation management systems which are usually built through well-established, highly centralized processes. Debian has spawned more than 200 GNU/Linux operating systems focused on non-profits, physicians, dentists, and Christians; none are demonstrably *better* than any of the others. They exist to serve the needs of small groups of users and are built by even smaller teams, usually 1–2 people — working with others in parallel . Without CIP, the vast majority of these operating systems would never have existed.

CIP for writing will lead to a new writing methodology that will lead to new types of communities and a new set of writing practices. The result will be texts that were not possible before and are only possible through CIP and facilitated by new CIP technologies.

Chapter 5

Related Technologies

There are hundreds of relevant collaborative writing tools that over the past several decades have helped inform and direct my work. In the interest of space, I will briefly discuss the four most important classes of collaborative work tools and discuss a representative example of each class:

1. Distributed source control management (SCM) tools represented by *Bazaar*;
2. Source code merge tools and merge modes represent by the *GNU Emacs* programming editor's *EDiff Merge Mode*;
3. Wikis represented by *Mediawiki*;
4. Word-processor change tracking represented by *Microsoft Word*;

Each example has been selected because it is representative of class of tools and I have tried to limit my analysis to traits that exist through the class. In my discussion, I will attempt to focus on the tools' abilities to support the five enumerated principles of CIP as laid out in Section 3.4.

5.1 Distributed Source Control Management (Bazaar)

In Section 3.3, I discussed DSCM technologies as an example of CIP for software development. With that introduction, DSCM tools seem like a natural place to begin a process of evaluating

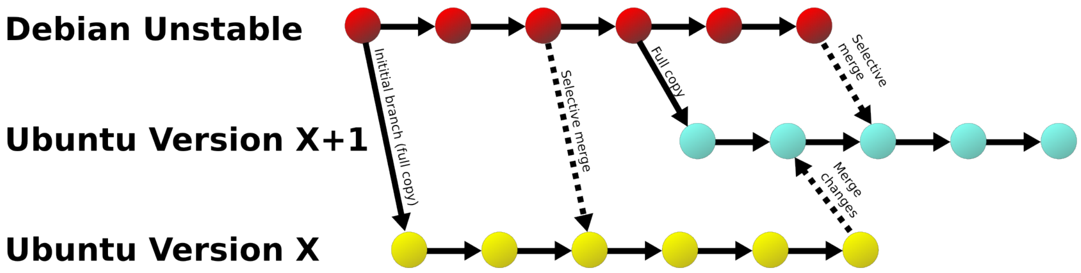


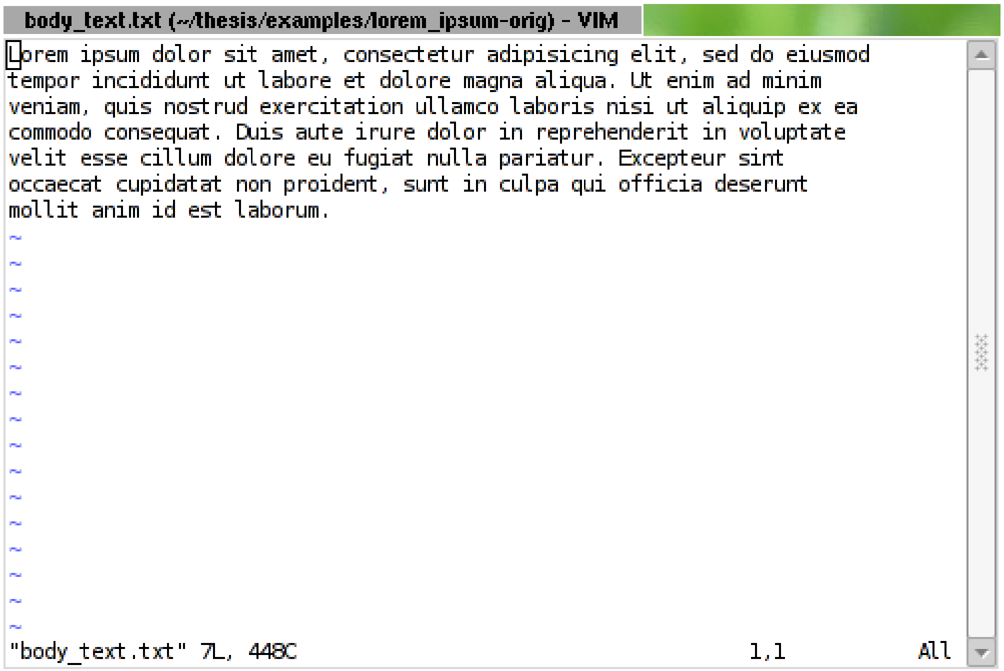
Figure 5-1: Visualization of Debian and Ubuntu's parallel development relationship (Adapted from Scott James Remnant)

technologies for their ability to support CIP for writing. For this analysis, I will focus on the *Bazaar* DSCM system.

Bazaar is a DSCM written by Canonical Limited to facilitate more effective development of the Ubuntu GNU/Linux distribution in parallel with the Debian GNU/Linux project. A model of the Ubuntu release cycle and its relationship to Debian is shown in Figure 5-1. As can be seen in the figure, Ubuntu development began with a pristine branch of Debian's "unstable" distribution. Ubuntu then proceeded to create a series of modifications to Unstable. This set of differences is carried over to new pristine copies of Debian at the beginning of each new Ubuntu release cycle. In this way, the Ubuntu development model is highly dependent on merging bug fixes, new features and user interface improvements made to Ubuntu into new versions of software in Debian. Ubuntu releases once every six months and is supported by a team of under 25 paid developers. As a result, the model is highly reliant on DSCM as a means of reducing the cost of maintaining and merging the code-level differences between the two projects [35].

Ubuntu's development model, facilitated by DSCM, provides an archetype for CIP. However, while DSCM tools may satisfy the five design principles for source code as in Ubuntu's case, they fail to do so for writing. Existing DSCM tools prove inappropriate for text because they are built around several central design decisions that, while highly appropriate for source code, render the tools problematic writers.

The most fundamental problem with the use of DSCM systems for text is the systems' strong emphasis on lines as the fundamental atomic unit for representing differences. Lines are an appropriate unit for source code because code is frequently understood, interacted, with, and measured



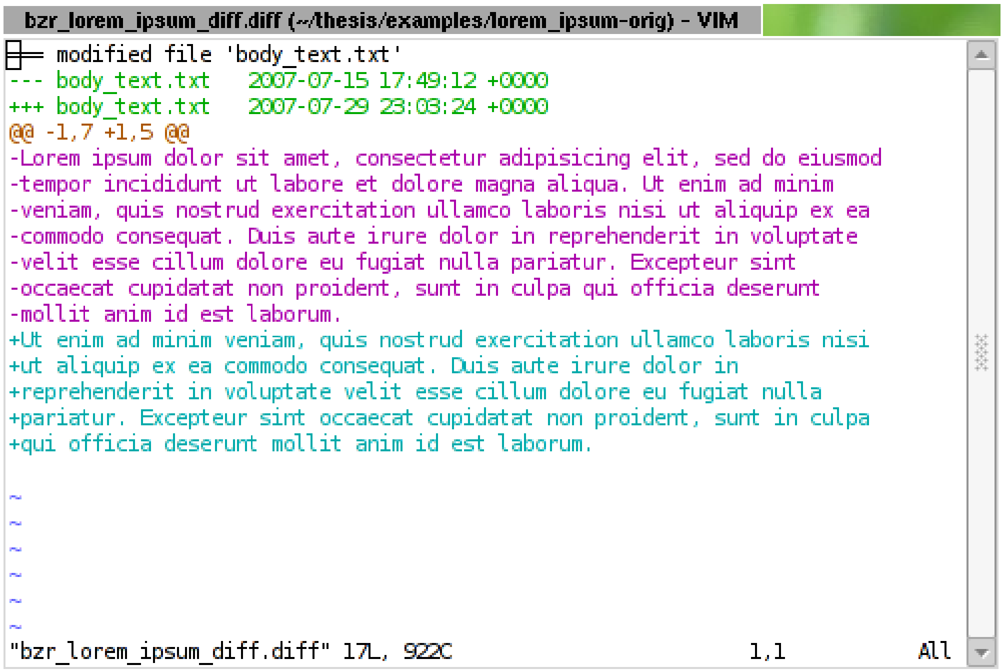
```
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat. Duis aute irure dolor in reprehenderit in voluptate
velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt
mollit anim id est laborum.
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"body_text.txt" 7L, 448C                                  1,1           All
```

Figure 5-2: File containing lorem ipsum text

by lines [54, 61]. Expressions in source code are designed to exist on a single line — a practice that is enforced in some programming languages like Python [72]. Representing differences in lines makes sense because even a small change to a line of source code will represent a changed expression that will require human reconsideration of the entire expression.

However, line-based approaches to representing differences is cumbersome and inappropriate for writing text. Take, for example, the paragraph made up of the standard *lorem ipsum* text as shown in Figure 5-2. Many wikis and WYSIWYM[1] writing applications will internally represent each paragraph as a single line. As a result, any change to a paragraph will result in a changed line with no indication to the user as to which part of the line differs. This situation does not change, however, if line breaks are taken into account. The "diff" from Bazaar of two paragraphs, the first with the first sentence removed is shown in Figure 5-3. New lines are prefixed with "+" and removed lines are prefixed with "-". Removing the first sentence sets off a chain reaction that change each line break in the paragraph. Because each line, taken individually, is different, the

[1] WYSIWYM stands for "What You See Is What You Mean" and is a term used in the LyX and LaTeX communities to refer to a class of WYSIWYG or almost WYSIWYG application [39].


bzr_lorem_ipsum_diff.diff (~/thesis/examples/lorem_ipsum-orig) - VIM

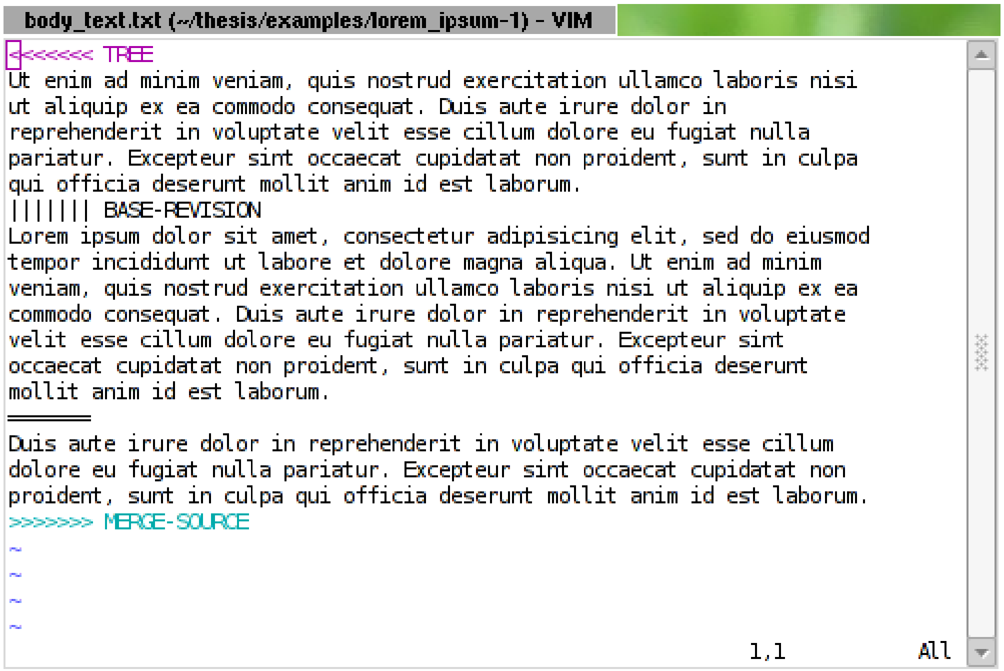
```
=== modified file 'body_text.txt'
--- body_text.txt   2007-07-15 17:49:12 +0000
+++ body_text.txt   2007-07-29 23:03:24 +0000
@@ -1,7 +1,5 @@
-Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
-tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
-veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
-commodo consequat. Duis aute irure dolor in reprehenderit in voluptate
-velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
-occaecat cupidatat non proident, sunt in culpa qui officia deserunt
-mollit anim id est laborum.
+Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi
+ut aliquip ex ea commodo consequat. Duis aute irure dolor in
+reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
+pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa
+qui officia deserunt mollit anim id est laborum.

~
~
~
~
~
~
"bzr_lorem_ipsum_diff.diff" 17L, 922C                        1,1          All
```

Figure 5-3: File containing the "diff" of the lorem ipsum text with a version without the first sentence

entire text is marked as having changed and users must find any actual divergence by hand.

Conflict resolution tools rely on "diffing" so problems associated with line-by-line difference representation are inherited and aggravated in conflict resolution within DSCM tools. In the following example, there are conflicting changes in two related branches. The first branch (called TREE) contains the *lorem ipsum* text with the first sentenced removed. The second branch (called MERGE-SOURCE) is the branch from which the changes are being merged. In the example, this branch contains the *lorem ipsum* text with both the first and second sentences extricated. Bazaar's merge functionality takes into account the history of the branch by referring to the two branches' common ancestor (called BASE-REVISION) to compute the differences. The resulting conflict representation is shown in the text of the file as shown in Figure 5-4.

The line-by-line method of conflict representation makes it difficult for writers to understand exactly what has changed. Additionally, DSCM output is non-interactive and provides neither hints nor methods for conflict resolution. Instead, DSCM users must hand-edit the conflict representation using an external text editor before marking the conflict as resolved. This historic



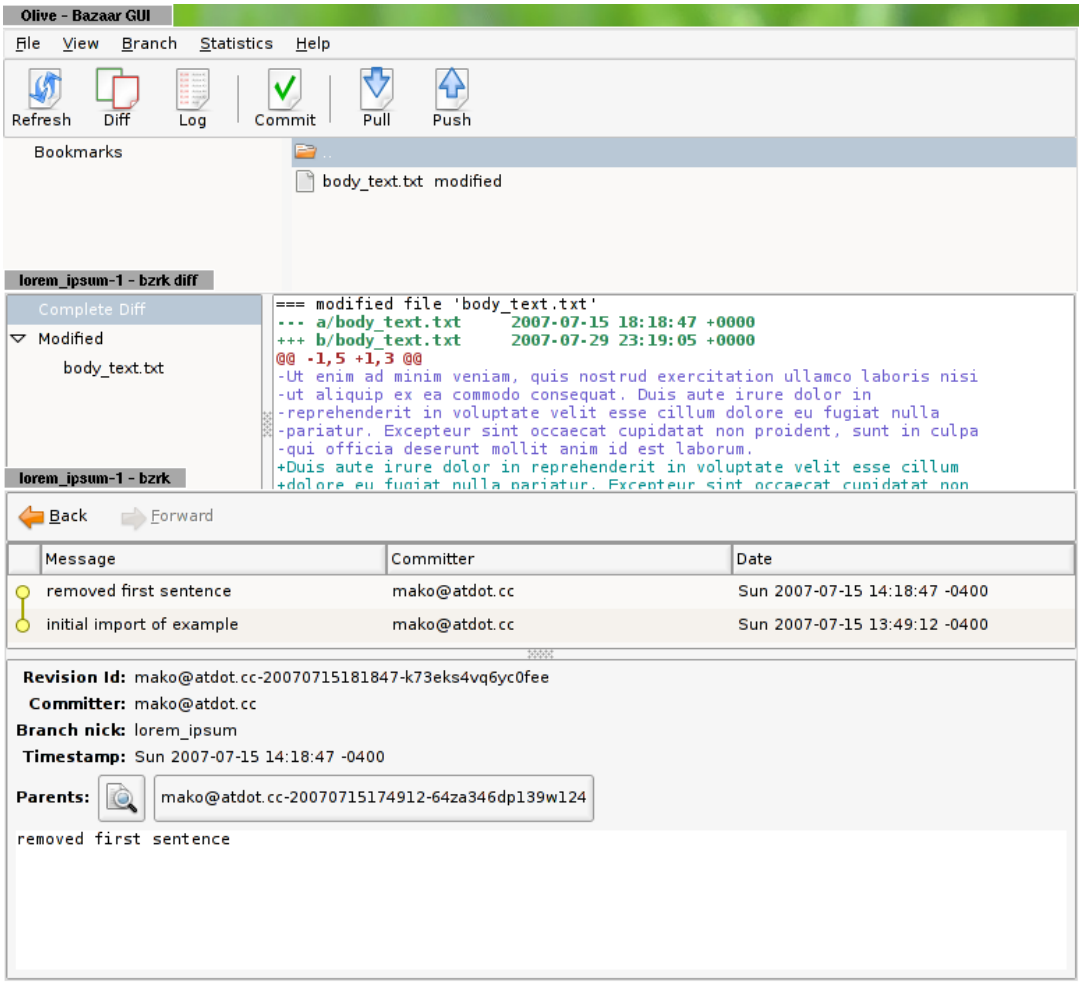
```
<<<<<<< TREE
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi
ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa
qui officia deserunt mollit anim id est laborum.
||||||| BASE-REVISION
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat. Duis aute irure dolor in reprehenderit in voluptate
velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt
mollit anim id est laborum.
=======
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum
dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
>>>>>>> MERGE-SOURCE
~
~
~
~
                                                        1,1         All
```

Figure 5-4: File containing the representation of a conflict after a merge in Bazaar

separation between DSCM systems and editors is designed to support modularity considered important in software development but creates a difficult barrier for writers.

Additional shortcomings in DSCM when used for documents include the lack of methods to allows users to become aware of other collaborators; a requirement that all managed documents be plain text; and command-line interfaces unfamiliar and prohibitively complex for many non-programmers.[2] *Olive*, a graphical front-end to Bazaar shown in Figure 5-5, attempts to provide a more approachable interface. However, its close functional mapping to Bazaar commands and its use of Bazaar's line-by-line diff output is still highly geared toward programmers and confusing for many less-technically skilled authors.

[2] Bazaar — famous for its relatively simple interface — has 91 sub-commands (each with several options and modifiers) and 14 sub-commands that can be considered essential for all users to know.



body_text.txt modified

body_text.txt

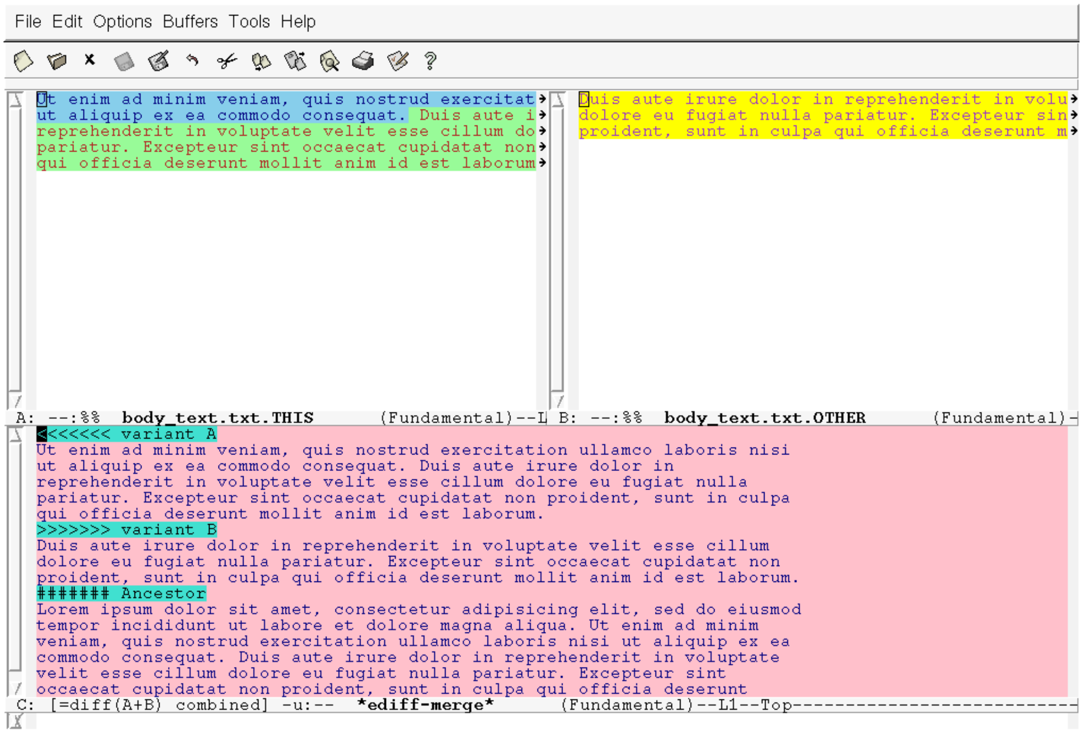
```
=== modified file 'body_text.txt'
--- a/body_text.txt     2007-07-15 18:18:47 +0000
+++ b/body_text.txt     2007-07-29 23:19:05 +0000
@@ -1,5 +1,3 @@
-Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi
-ut aliquip ex ea commodo consequat. Duis aute irure dolor in
-reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
-pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa
-qui officia deserunt mollit anim id est laborum.
+Duis aute irure dolor in reprehenderit in voluptate velit esse cillum
+dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
```

Message	Committer	Date
removed first sentence	mako@atdot.cc	Sun 2007-07-15 14:18:47 -0400
initial import of example	mako@atdot.cc	Sun 2007-07-15 13:49:12 -0400

Revision Id: mako@atdot.cc-20070715181847-k73eks4vq6yc0fee
Committer: mako@atdot.cc
Branch nick: lorem_ipsum
Timestamp: Sun 2007-07-15 14:18:47 -0400
Parents: mako@atdot.cc-20070715174912-64za346dp139w124

```
removed first sentence
```

Figure 5-5: Screenshot of the *Olive* front-end to Bazaar



```
Ut enim ad minim veniam, quis nostrud exercitat
ut aliquip ex ea commodo consequat. Duis aute i
reprehenderit in voluptate velit esse cillum do
pariatur. Excepteur sint occaecat cupidatat non
qui officia deserunt mollit anim id est laborum
```

```
Duis aute irure dolor in reprehenderit in volu
dolore eu fugiat nulla pariatur. Excepteur sin
proident, sunt in culpa qui officia deserunt m
```

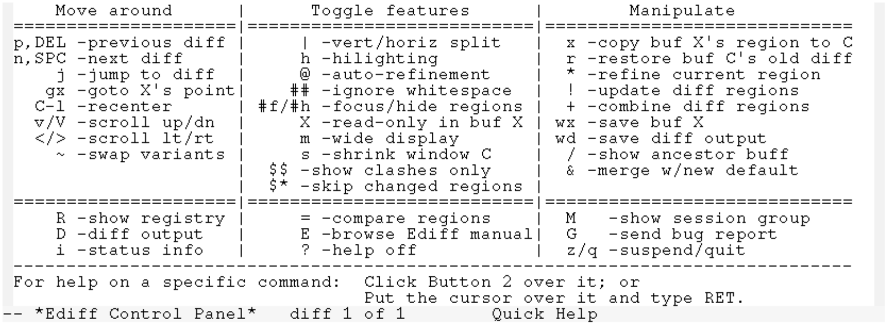
```
A: --:%%  body_text.txt.THIS       (Fundamental)--L B: --:%%  body_text.txt.OTHER       (Fundamental)-
<<<<<<< variant A
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi
ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa
qui officia deserunt mollit anim id est laborum.
>>>>>>> variant B
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum
dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
####### Ancestor
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat. Duis aute irure dolor in reprehenderit in voluptate
velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt
C: [=diff(A+B) combined] -u:--  *ediff-merge*     (Fundamental)--L1--Top---------------------------
```

Figure 5-6: Screenshot of the *Emacs EDiff* main window

5.2 Source Merge Tools and Modes (Emacs)

Not only are conflict resolution methods in DSCM system ill-suited to naive adaption to writing, they are notoriously difficult for software developers [45]. As a result, programmers have created tools for automating merge processes. In most cases, these tools are designed to be used in conjunction with SCM or DSCM tools. Their goal is to represent, reflect, and provide interfaces to resolve conflicts during merges of source code. Often, these take the form of specialized merge tools like the *Guiffy Merge Tool* or the *Perforce Merge Tool*, Ellie Computing's *ECMerge* and Mikado Limited's *MergePlanet*. In other cases, merge tools are integrated into programming editors as "merge modes." Well respected merge tools are built into the Eclipse and GNU Emacs programming editors. Due to its longevity and reputation, I will analyze Emacs's merge mode in this section.

Emacs supports both two- and three-way merges and can support simple history sensitive merg-



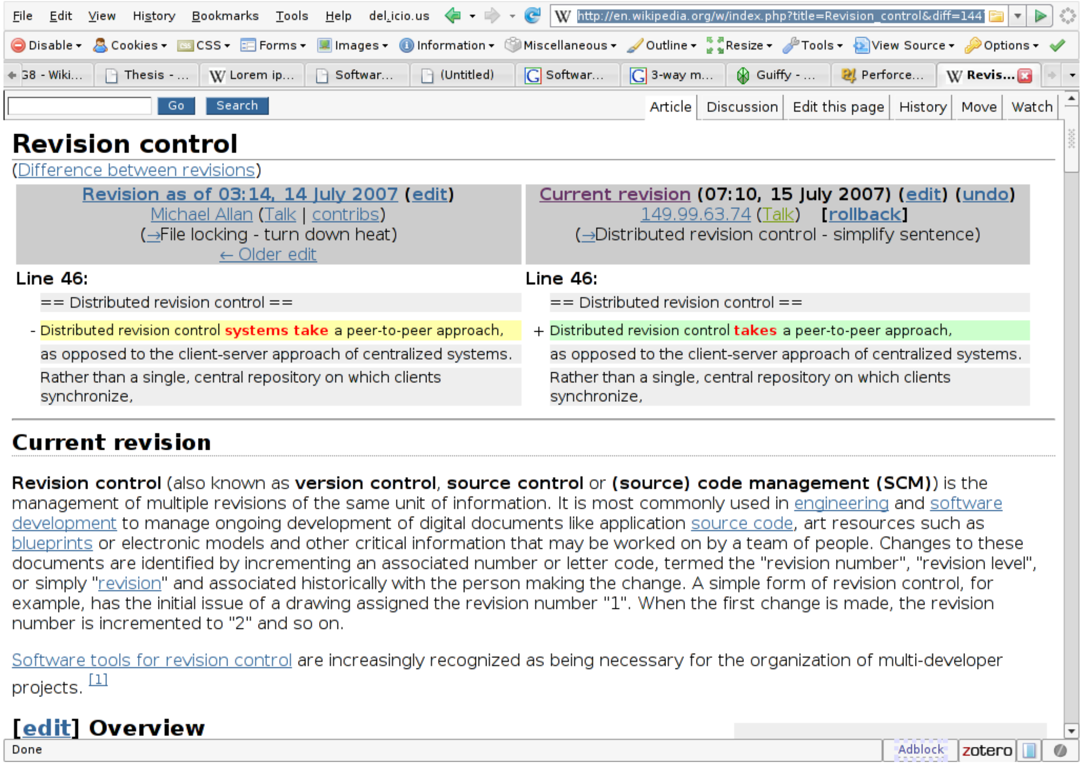
```
    Move around      |      Toggle features      |        Manipulate
=====================|===========================|=============================
p,DEL -previous diff |      | -vert/horiz split   |  x -copy buf X's region to C
n,SPC -next diff     |      h -hilighting         |  r -restore buf C's old diff
    j -jump to diff  |      @ -auto-refinement    |  * -refine current region
   gx -goto X's point|     ## -ignore whitespace  |  ! -update diff regions
  C-l -recenter      | #f/#h -focus/hide regions  |  + -combine diff regions
  v/V -scroll up/dn  |      X -read-only in buf X | wx -save buf X
  </> -scroll lt/rt  |      m -wide display       | wd -save diff output
    ~ -swap variants |      s -shrink window C    |  / -show ancestor buff
                     |    $$ -show clashes only   |  & -merge w/new default
                     |    $* -skip changed regions |
=====================|===========================|=============================
    R -show registry |      = -compare regions    |  M   -show session group
    D -diff output   |      E -browse Ediff manual|  G   -send bug report
    i -status info   |      ? -help off           |  z/q -suspend/quit
-------------------------------------------------------------------------------
For help on a specific command:  Click Button 2 over it; or
                                 Put the cursor over it and type RET.
-- *Ediff Control Panel*   diff 1 of 1        Quick Help
```

Figure 5-7: Screenshot of the *Emacs EDiff* control panel

ing with its built in `ediff-merge-with-ancestor` function. The output of that command is the three-paned view shown in Figure 5-6. It consists of a side-by-side comparison of the two diverged branches. In the third, larger pane at the bottom of of the window is the merged version containing a conflict representation similar to the output of DSCM as shown in Figure 5-4. This window is used in conjunction with a control panel like the one shown in Figure 5-7 that allows users to use their keyboard to navigate through changes and to select or move changes from either branch into the merged document in the bottom pane.

When used for writing, merge tools like the Emacs' merge mode suffer from the same same fundamental problem as DSCM by relying on lines as the fundamental atomic unit. This drawback is complicated by a two-pane, side-by-side view that is optimized for showing differences between changed lines. When used for writing, this system is poorly suited to keeping different panes in sync and helping authors to contextualize changes.

However, merge tools like Emacs offer one distinct advantage over DSCM alone by computing word-by-word diffs *within* changed regions of lines; the blue highlighted text in Figure 5-6 is a representative example. Additionally, when several lines in a large file are changed, merge modes provide a mechanism to step through a large file by jumping immediately to the next changed "hunk" of changed lines and by allowing users to easily select the version from either of the branches.


File Edit View History Bookmarks Tools Help del.icio.us
http://en.wikipedia.org/w/index.php?title=Revision_control&diff=144
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Source Options
58 - Wiki... Thesis - ... Lorem ip... Softwar... (Untitled) Softwar... 3-way m... Guiffy - ... Perforce... Revis...
Go
Search
Article
Discussion
Edit this page
History
Move
Watch
Done
Adblock
zotero

Revision control

(Difference between revisions)

Revision as of 03:14, 14 July 2007 (edit) Michael Allan (Talk | contribs) (→File locking - turn down heat) ← Older edit	Current revision (07:10, 15 July 2007) (edit) (undo) 149.99.63.74 (Talk) [rollback] (→Distributed revision control - simplify sentence)
Line 46:	**Line 46:**
== Distributed revision control ==	== Distributed revision control ==
- Distributed revision control **systems take** a peer-to-peer approach,	+ Distributed revision control **takes** a peer-to-peer approach,
as opposed to the client-server approach of centralized systems.	as opposed to the client-server approach of centralized systems.
Rather than a single, central repository on which clients synchronize,	Rather than a single, central repository on which clients synchronize,

Current revision

Revision control (also known as **version control**, **source control** or **(source) code management (SCM)**) is the management of multiple revisions of the same unit of information. It is most commonly used in engineering and software development to manage ongoing development of digital documents like application source code, art resources such as blueprints or electronic models and other critical information that may be worked on by a team of people. Changes to these documents are identified by incrementing an associated number or letter code, termed the "revision number", "revision level", or simply "revision" and associated historically with the person making the change. A simple form of revision control, for example, has the initial issue of a drawing assigned the revision number "1". When the first change is made, the revision number is incremented to "2" and so on.

Software tools for revision control are increasingly recognized as being necessary for the organization of multi-developer projects. [1]

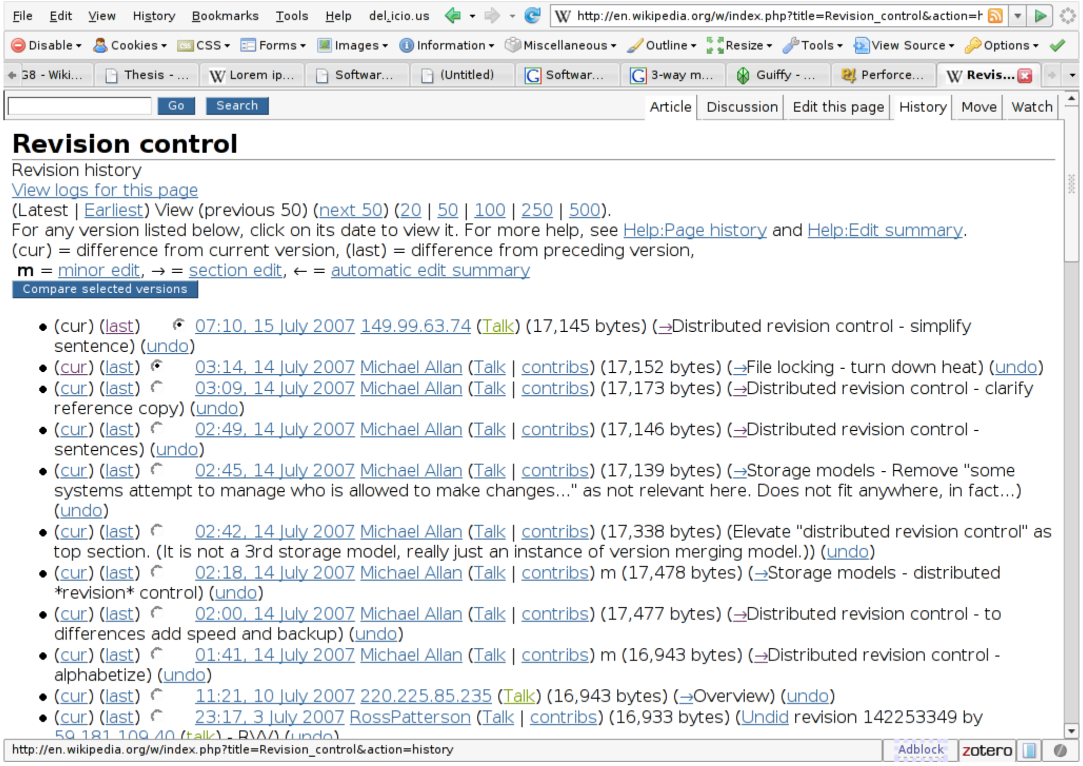
[edit] Overview

Figure 5-8: Example "diff" of a set of changes between two versions of wiki page in Wikipedia running the *Mediawiki* wiki software

5.3 Wikis (Mediawiki)

The first wiki, written by Ward Cunningham, was designed to be an easy way to update pages on the web. Soon after, wikis incorporated basic RCS-style SCM features [44]. On modern wikis, every change to a page is saved as a new version and users can diff two revisions of a page and retrieve any earlier version. As a representative example, I evaluated Mediawiki: the wiki software that runs Wikipedia and many other wikis and perhaps the most featureful and popular wiki software [78].

Mediawiki's inspiration in SCM and merge tools is obvious in its two-pane diff page shown in Figure 5-8. Like SCM and DSCM tools, changes in wikis are represented as changed lines. Like merge tools, word-by-word differences are shown within the changed lines. Also like merge tools, a two-pane, side-by-side model is adopted. With strong emphasis on word-by-word changes


File Edit View History Bookmarks Tools Help del.icio.us
http://en.wikipedia.org/w/index.php?title=Revision_control&action=h
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Source Options
58 - Wiki... Thesis - ... Lorem ip... Softwar... (Untitled) Softwar... 3-way m... Guiffy - ... Perforce... Revis...
Go Search
Article Discussion Edit this page History Move Watch

Revision control

Revision history

View logs for this page

(Latest | Earliest) View (previous 50) (next 50) (20 | 50 | 100 | 250 | 500).

For any version listed below, click on its date to view it. For more help, see Help:Page history and Help:Edit summary.

(cur) = difference from current version, (last) = difference from preceding version,
m = minor edit, → = section edit, ← = automatic edit summary

Compare selected versions

- (cur) (last) ● 07:10, 15 July 2007 149.99.63.74 (Talk) (17,145 bytes) (→Distributed revision control - simplify sentence) (undo)
- (cur) (last) ● 03:14, 14 July 2007 Michael Allan (Talk | contribs) (17,152 bytes) (→File locking - turn down heat) (undo)
- (cur) (last) ○ 03:09, 14 July 2007 Michael Allan (Talk | contribs) (17,173 bytes) (→Distributed revision control - clarify reference copy) (undo)
- (cur) (last) ○ 02:49, 14 July 2007 Michael Allan (Talk | contribs) (17,146 bytes) (→Distributed revision control - sentences) (undo)
- (cur) (last) ○ 02:45, 14 July 2007 Michael Allan (Talk | contribs) (17,139 bytes) (→Storage models - Remove "some systems attempt to manage who is allowed to make changes..." as not relevant here. Does not fit anywhere, in fact...) (undo)
- (cur) (last) ○ 02:42, 14 July 2007 Michael Allan (Talk | contribs) (17,338 bytes) (Elevate "distributed revision control" as top section. (It is not a 3rd storage model, really just an instance of version merging model.)) (undo)
- (cur) (last) ○ 02:18, 14 July 2007 Michael Allan (Talk | contribs) m (17,478 bytes) (→Storage models - distributed *revision* control) (undo)
- (cur) (last) ○ 02:00, 14 July 2007 Michael Allan (Talk | contribs) (17,477 bytes) (→Distributed revision control - to differences add speed and backup) (undo)
- (cur) (last) ○ 01:41, 14 July 2007 Michael Allan (Talk | contribs) m (16,943 bytes) (→Distributed revision control - alphabetize) (undo)
- (cur) (last) ○ 11:21, 10 July 2007 220.225.85.235 (Talk) (16,943 bytes) (→Overview) (undo)
- (cur) (last) ○ 23:17, 3 July 2007 RossPatterson (Talk | contribs) (16,933 bytes) (Undid revision 142253349 by 59.181.109.40 (talk) - RVV) (undo)

http://en.wikipedia.org/w/index.php?title=Revision_control&action=history
Adblock
zotero

Figure 5-9: History page for an article in Wikipedia running the *Mediawiki* wiki software

within regions, Mediawiki and other wikis offer document comparison functionality that is more useful than source-based merge modes for representing differences between documents.

However, neither Mediawiki nor any other wiki has any support for branching or merging pages. Like the older SCM tools that DSCM aimed to improve upon, wikis treat history as entirely serial — as is apparent in Mediawiki's history view as shown in Figure 5-9. Like centralized SCM tools, wikis work without support for branches because they can easily impose a requirement that users interact with a central server — the web's standard interaction model. As a result, wikis provide ways to revert or replay changes but no support for parallel authoring. Wiki users are free to create parallel texts by cutting and pasting content but, with no model for representing branches within the wiki, provide no technical mechanisms to help authors keep diverged branches synchronized.

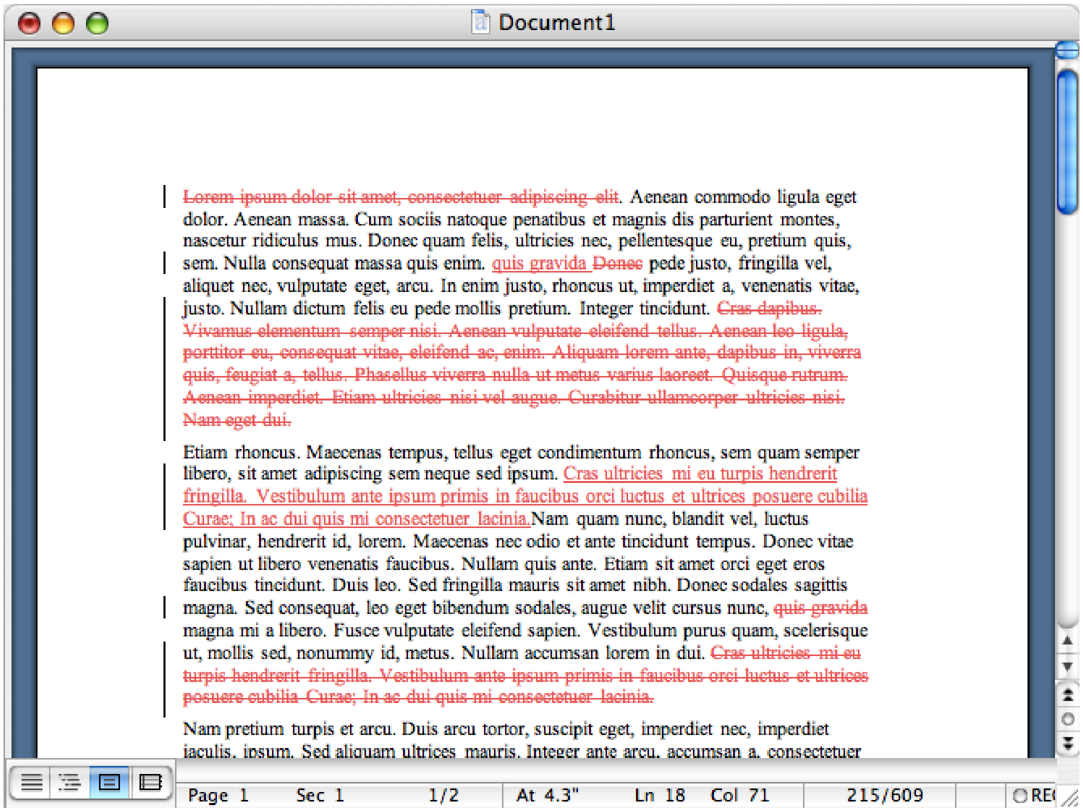
By providing a web browser-based interface, Mediawiki and other wikis have successfully lowered the barrier to contribution and succeeded in providing a unified, consistent interface for collaboratively authoring and publishing. However, while wikis have integrated ideas and technologies from SCM, they have done little to import ideas from the latest generation of distributed DSCM tools. While excellent at supporting collaboration, wikis are poor at supporting collaboration in parallel. The result is less divergence and, ultimately, a failure of wikis to reach their full potential.

5.4 Track Changes (Microsoft Word)

A final technology for evaluation is the track changes interfaces built into the popular word processors Microsoft Word and OpenOffice.org Writer. Designed originally for non-collaborative work, the pervasive use of word processors in collaborative settings has lead to the inclusion of many features designed to support cooperative writing. The most important of these in the context of this analysis are the features related to tracking, recording, and representing changes.

Built around use by individual authors, Word provides no support for centralized repositories. As a result, it allows parallel work on any copy of any document. While this opens the door to a rich environment for CIP, Word provides little support for merging the resulting diverged changes. This means that in many cases of document divergence with conflicting changes, any divergence must be computed by hand through a side-by-side comparison of the complete document by a human or, if the documents have diverged only slightly, through a simple two two-way diff feature. Word emphasizes transparency by offering little integrated support for workflow. However, the cost of merging any divergence is so high in the Word model that to be effective the system requires it. Like wikis, Word serializes changes into a linear history and provides no internal model for tracking branches.

Because document comparison functionality is limited, the most reliable methods for collaboratively using Word's track changes functionality is to impose strong workflow. Frequently, this happens through the use of a locking revision control system like RCS that ensures that only one user can edit a document at a particular time. Examples include Microsoft's SharePoint [23]. SharePoint, and similar Word-based systems, have no support for branching.



~~Lorem ipsum dolor sit amet, consectetuer adipiscing elit~~. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. quis gravida ~~Donec~~ pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt. ~~Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim. Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus. Phasellus viverra nulla ut metus varius laoreet. Quisque rutrum. Aenean imperdiet. Etiam ultricies nisi vel augue. Curabitur ullamcorper ultricies nisi. Nam eget dui.~~

Etiam rhoncus. Maecenas tempus, tellus eget condimentum rhoncus, sem quam semper libero, sit amet adipiscing sem neque sed ipsum. Cras ultricies mi eu turpis hendrerit fringilla. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; In ac dui quis mi consectetuer lacinia.Nam quam nunc, blandit vel, luctus pulvinar, hendrerit id, lorem. Maecenas nec odio et ante tincidunt tempus. Donec vitae sapien ut libero venenatis faucibus. Nullam quis ante. Etiam sit amet orci eget eros faucibus tincidunt. Duis leo. Sed fringilla mauris sit amet nibh. Donec sodales sagittis magna. Sed consequat, leo eget bibendum sodales, augue velit cursus nunc, ~~quis gravida~~ magna mi a libero. Fusce vulputate eleifend sapien. Vestibulum purus quam, scelerisque ut, mollis sed, nonummy id, metus. Nullam accumsan lorem in dui. ~~Cras ultricies mi eu turpis hendrerit fringilla. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; In ac dui quis mi consectetuer lacinia.~~

Nam pretium turpis et arcu. Duis arcu tortor, suscipit eget, imperdiet nec, imperdiet iaculis, ipsum. Sed aliquam ultrices mauris. Integer ante arcu, accumsan a, consectetuer

Figure 5-10: Screenshot of *Microsoft Word* showing changes made to a document

Word processors distinguish themselves from wikis and merge tools by offering truly text-specific interfaces to record and reflect changes. Word uses colors to represent collaborators and reflects changes in an in-line, character-by-character manner, as shown in Figure 5-10. This approach to difference presentation is compact, contextualized, and text-specific.

5.5 Building on Related Work

DSCM goes a long way toward supporting CIP for writing by implementing algorithms and methods for history-sensitive merging and work in parallel branches. However, DSCM falls short of providing adequate tools for writers due to structural elements and interfaces that, while appropriate for writers of source code, are problematic or confusing for writers of text. Merge modes provide an interaction style that makes merging source code easier but, like DSCM, offers interfaces that are substantially more useful for software developers than for writers. While wikis make overtures towards text-appropriate interfaces and succeed in using web interfaces to facilitate widespread collaboration that is fertile soil for CIP, they do so at the cost of all branching and merging functionality. They do, however, Finally, word processors offer effective document-specific interfaces to tracking, recording, and presenting changes but, like wikis, do so without any support for parallel work. While none of these tools support CIP for writing, they each demonstrate concepts that inform the design of a better system. Taken together, these tools provide most of the components necessary for a novel, document specific interface for supporting collaboration in parallel.

Chapter 6

Design

TextNet is a system designed to support collaboration in parallel for text. While interaction with TextNet involves simple wiki-like operation when used alone, an instance of TextNet is designed to interact with other instances to support collaborative work through parallel maintenance of documents. TextNet builds on the concepts and theories described in Chapter 3 and the technologies described in Chapter 5 including DSCM, merge modes, wikis, and in-line word-by-word diffs to build a new system that facilitates CIP for writing. The design goals for the system can fall roughly in four major categories:

1. *Branch accounting*: Finding out what relatives exist and when they have changed;
2. *Difference representation*: Effectively representing changes to text on the web;
3. *Conflict resolution*: Presenting an effective interface for computing, displaying, and helping users resolve conflicts upon merges;
4. *Wiki interface*: Making an interface work effectively in a web and wiki interface.

These criteria are directly motivated by and adapted from the theoretical design criteria for CIP tools described in Section 3.4 but adapted to the particularities of collaborative writing. Theoretical criteria one and two — support for parallel work interspersed by merges and *ad hoc* style

— are a core description for CIP's method of work and frame and motivate each of the technical design criteria listed above. They serve as a higher level description of the purpose, as opposed to a set of testable technical design criteria. Criteria one aims to ensure support transparency — theoretical criteria two — and allows collaborators to quickly and easily decide when and who to work with — theoretical criteria five. Criteria two is a restatement of theoretical criteria three — effectively displaying and summarizing changes between documents. Criteria three, in turn, is a restatement of theoretical criteria four — representing, reflecting, and resolving conflicts upon merges.

Criteria four is a meta requirement: while several of the design criteria emphasize effectiveness and appropriateness of interfaces, the requirement of a wiki-like interface introduces a design requirement that aims to introduce an interface that is effective and appropriate for collaborative writing. Wikis are already one of the most high-profile spaces for *ad hoc* collaboration on the web. The adoption of a wiki interface helps satisfy theoretical criteria two while helping support an accessible and effective model for each of the theoretical design goals.

6.1 Branch Accounting

While DSCM tools all create branches and store and merge changes between branches, no widely used DSCM system maintains a list of related branches and the status of those relatives. Users of DSCM tools are expected to merge changes between branches when prompted to by "out of band" discussions like email. Bazaar has built-in support for "bound branches" which support a parent-child relationship. Use of bound branches allows for a single one-level star topology of branches where each branch node is connected directly to the central parent branch and to no other branches as shown in Figure 6-1: to move a change from B to C would require that the change be merged into A from B first before it is merged into C through A. While repositories and history are distributed in Bazaar bound branches, the collaboration model is identical to work under a centralized system; users can only see changes from other leaf nodes when they are brought *through* the center. The inconveniences of this situation are aggravated in a more complex star topology as shown in Figure 6-2. For branch E to see a change by D it would need

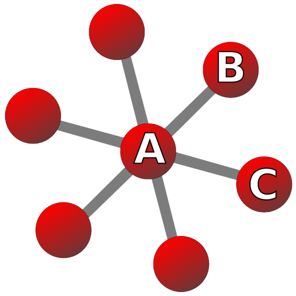


Figure 6-1: Bazaar branches in a simple "star" topology

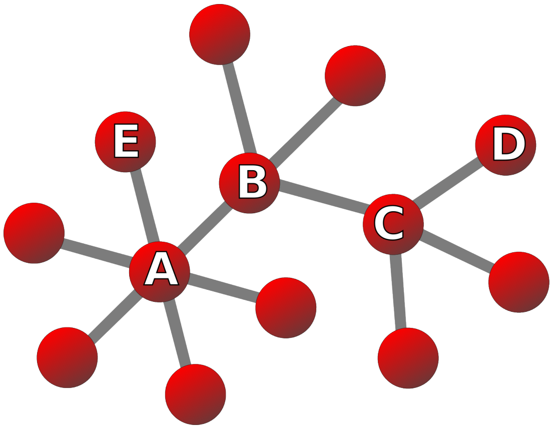


Figure 6-2: Bazaar branches in a more complex topology of "nested" stars

to be merged by C, B, and A — in that order. The result, in most cases, is workflow designed to support the hierarchy.

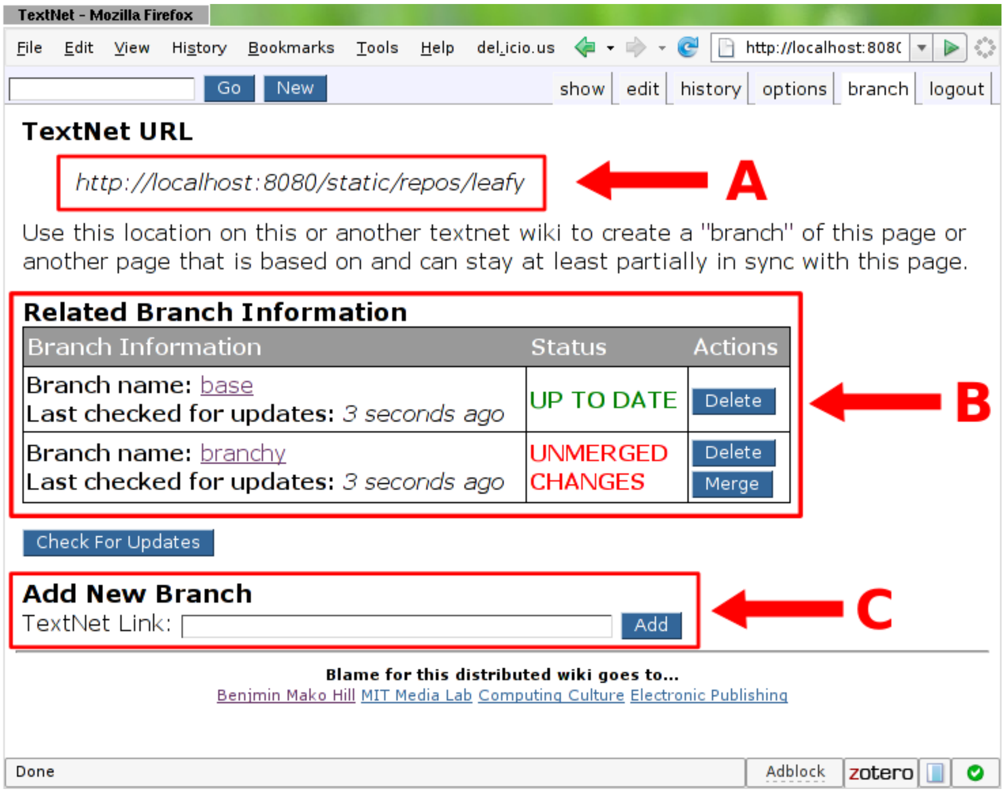
In CIP, changes must be able to be freely shared without in-depth knowledge of the network of collaborators and should be able to create an inter-linked network of individuals. For example, Author C in Figure 6-2 should be able to branch from Author B who may branch from Author A; Additionally, authors A and C should be able to share changes directly and should be able to collaborate with Author B or D in the future without risk of spurious conflicts. By supporting this mode of work in an *ad hoc* manner, TextNet allows for relationships unmediated by workflow.

For the purpose of merging in a DSCM system, it is important to identify a common ancestor. In GNU Arch, the original FOSS DSCM system, merge algorithms required a star network topology to ensure that there was a common ancestor. For example, merging D to E in Figure 6-2 would result in spurious conflicts because changes might seem be coming from both places and the system would have no way to note that changes were shared between the two. Modern systems, like Bazaar, are able to represent common ancestors as the sum of all changesets shared between any two branches without regard to the branches' relationship. In other words, a common ancestor need not exist as a revision in an extant branch if it can be represented internally during the merge process. TextNet employs this second method of "graph merging," which eliminates the need for workflow to maintain a star topology and allows for the type of *ad hoc* group formation essential to CIP. In Figure 6-2, changes could be merged directly between D and E or between D and B without first passing through C.

The only prerequisite for this method of merging is that repositories and changesets must be uniquely identified. In TextNet, each page or article is stored as a Bazaar repository along with several files outside of the SCM versioning that store essential metadata. TextNet uses Bazaar to ensure that changesets are uniquely identified using a hash of the change itself combined with associated metadata. Because TextNet is a wiki, it can uniquely identify documents based on the URLs of the pages themselves. Each TextNet article comes with a "TextNet URL" visible on a special "Branch tab" — a special page associated with each article in a TextNet wiki that deals with branching related features — as shown in Figure 6-3, Area A.

Because TextNet computes common ancestors based on shared changesets, it does not need to track the relationships of related branches to each other. TextNet can create a list of shared changesets using the local and remote repositories themselves at the time of merging. For accounting purposes, TextNet merely needs to store the location of each related branch along side metadata, including when the branch was last synced and which changesets it contained at the time of the last merge. TextNet stores this data in a file in each article's associated SCM repository and uses this data to create the related branch list for each page shown in Figure 6-3, Area B.

When a new branch is created it can be either created *de novo* or can be created as a branch of an existing page as shown in Figure 6-4. To assist in the process of discovery of related branches, each branch created from an existing article will download the list of related branches from its



TextNet URL

http://localhost:8080/static/repos/leafy

Use this location on this or another textnet wiki to create a "branch" of this page or another page that is based on and can stay at least partially in sync with this page.

Related Branch Information

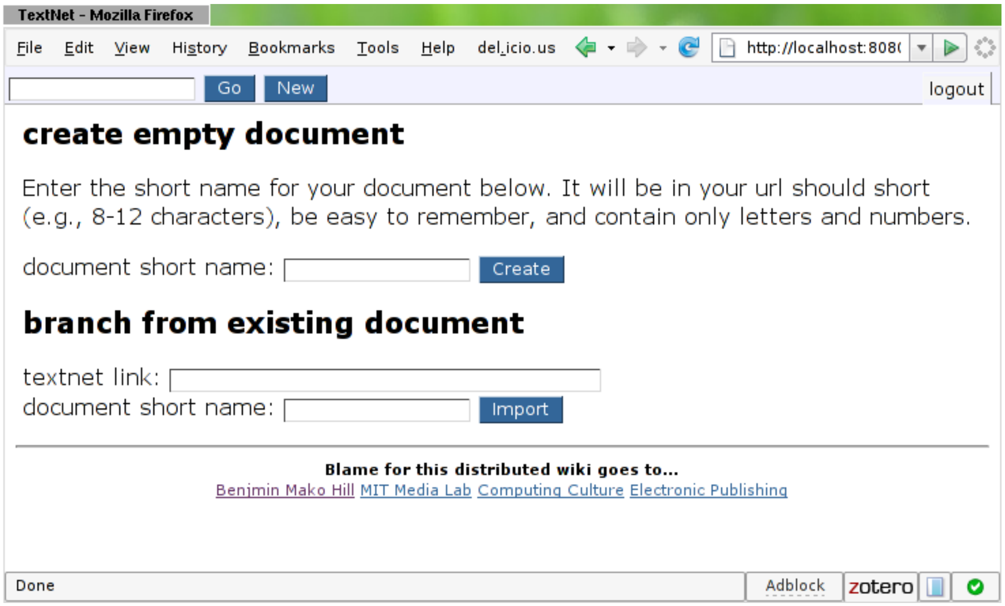
Branch Information	Status	Actions
Branch name: base **Last checked for updates:** *3 seconds ago*	UP TO DATE	Delete
Branch name: branchy **Last checked for updates:** *3 seconds ago*	UNMERGED CHANGES	Delete Merge

Add New Branch

TextNet Link:

Blame for this distributed wiki goes to...
Benjmin Mako Hill MIT Media Lab Computing Culture Electronic Publishing

Figure 6-3: Screenshot of TextNet's "Branch" tab



TextNet - Mozilla Firefox
File Edit View History Bookmarks Tools Help del.icio.us
http://localhost:808(
Go
New
logout

create empty document

Enter the short name for your document below. It will be in your url should short (e.g., 8-12 characters), be easy to remember, and contain only letters and numbers.

document short name: Create

branch from existing document

textnet link:
document short name: Import

Blame for this distributed wiki goes to...
Benjmin Mako Hill MIT Media Lab Computing Culture Electronic Publishing

Figure 6-4: Create new page in TextNet

parent when it is created. Additionally, it will "ping" its parent branch, which will, by default, record the child in the list of ancestors. At any time, users can explicitly add a relative by adding a new TextNet URL on the branch management screen as shown in Figure 6-3, Area C. As branches diverge, the number of differences and the effort required to resolve them increases and the relevancy decreases. When collaboration between two branches becomes too labor intensive, or less valuable, users can drop any branch from the list of relatives with the "Delete" button associated with each branch.

Users are able to prompt TextNet to synchronize with each related branch and determine if there are unmerged or unreviewed changes. This status is marked in the branch display window shown in Figure 6-3, Area B. Using this interface, users can merge changes from branches with unmerged changes at any time.

TextNet's branch discovery and accounting system is the first system of its kind. Its model is flexible, leaves the choice to collaborate or to diverge up to users, and does not enforce workflow. It is designed to expose information about what is going on in a way that invites, but does not enforce, collaboration in parallel. By emphasizing the role of collaborative work in related but

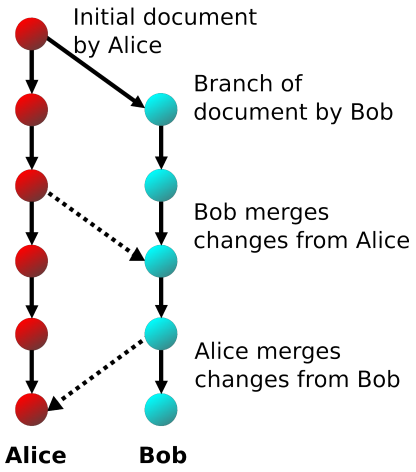

Initial document by Alice
Branch of document by Bob
Bob merges changes from Alice
Alice merges changes from Bob
Alice
Bob

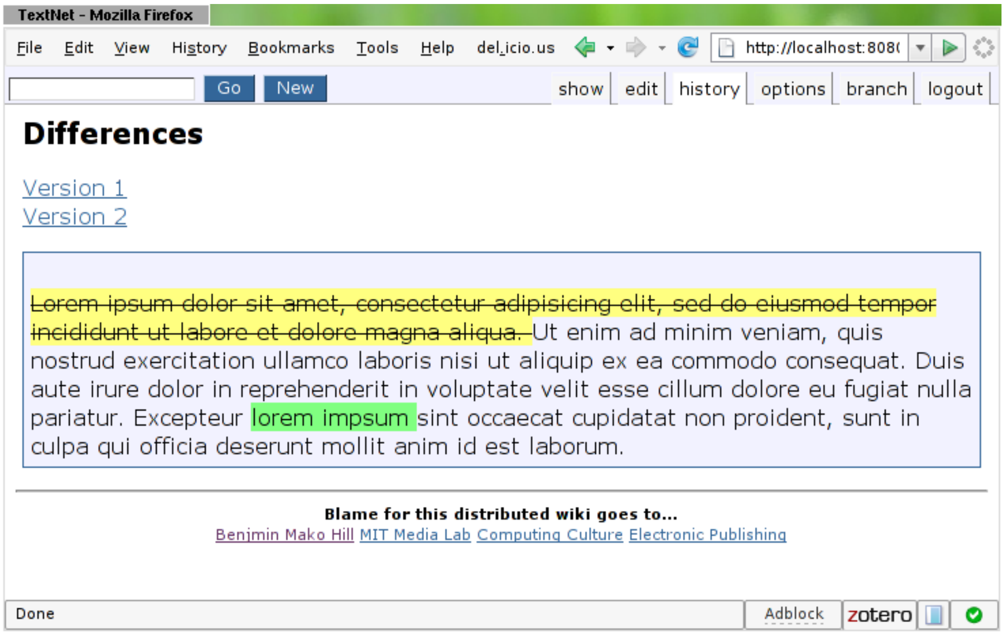
Figure 6-5: Visualization of two documents being worked on in parallel.

distinct branches, the branch accounting functionality in TextNet takes one major step toward supporting CIP for writing.

6.2 Difference Computation and Representation

Using the Bazaar merge library, TextNet makes extensive use of history-sensitive merging to minimize differences presented to users during merges. For example, if Bob branches from Alice and if Alice then makes a change that Bob does not want to integrate, Bob can merge with Alice but choose not to apply the changes in question. When Bob merges with Alice again later, TextNet will not simply show a difference between the latest copy of Bob's document and the latest copy of Alice's. Instead, it will show a set of differences that takes into account Bob's previous decision to diverge from in Alice in part. Figure 6-5 graphs out this example in detail. While the set of differences between the two documents at the time of the second merge contain the set of changes reviewed and rejected by Bob, it will be displayed. Differences are presented in ways that take into account previous choices to provide optimally concise lists of changes.

As discussed in Chapter 5, wikis and other software-inspired writing tools have struggled to move beyond difference representation methods optimized for source code that are often inefficient or inappropriate for writing. Rather than interacting with text as series of lines, documents in


TextNet - Mozilla Firefox
File Edit View History Bookmarks Tools Help del.icio.us
http://localhost:808(
Go New
show edit history options branch logout
Done Adblock zotero

Differences

Version 1
Version 2

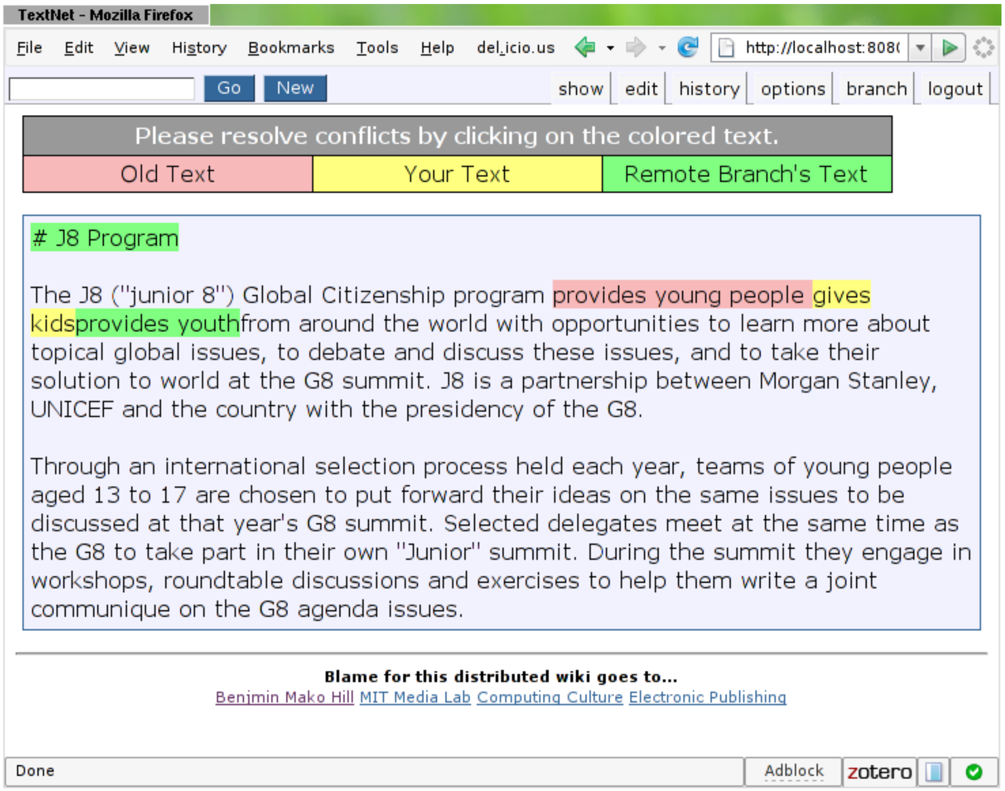
~~Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.~~ Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur lorem impsum sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Blame for this distributed wiki goes to...
Benjmin Mako Hill MIT Media Lab Computing Culture Electronic Publishing

Figure 6-6: Word-by-word "diff" in TextNet

TextNet are represented as paragraphs and words. Documents are navigated, displayed, and diffed first as paragraphs and then as words.

A simple example of the word-based diff change can be seen in Figure 6-6. Contrast this with the same change in Mediawiki which was shown in Figure 5-8 in Chapter 5. While word-based difference tools exist in Word, they are a novel addition to wikis. Additionally, TextNet uses this word-by-word diff model to support a novel word-based merge mode that offers significant improvements over the technologies shown in Chapter 5 Section 5.2 and which will be discussed in more depth below in Section 6.3 of this chapter. By easily and efficiently computing and representing changes, TextNet enhances transparency and makes parallel work increasingly possible. Appropriate difference representation methods like those described in this section offer another step toward bringing CIP for writing into reach.



Please resolve conflicts by clicking on the colored text.

Old Text	Your Text	Remote Branch's Text

J8 Program

The J8 ("junior 8") Global Citizenship program provides young people gives kidsprovides youthfrom around the world with opportunities to learn more about topical global issues, to debate and discuss these issues, and to take their solution to world at the G8 summit. J8 is a partnership between Morgan Stanley, UNICEF and the country with the presidency of the G8.

Through an international selection process held each year, teams of young people aged 13 to 17 are chosen to put forward their ideas on the same issues to be discussed at that year's G8 summit. Selected delegates meet at the same time as the G8 to take part in their own "Junior" summit. During the summit they engage in workshops, roundtable discussions and exercises to help them write a joint communique on the G8 agenda issues.

Blame for this distributed wiki goes to...
Benjmin Mako Hill MIT Media Lab Computing Culture Electronic Publishing

Figure 6-7: Displaying conflicts in TextNet

6.3 Conflict Resolution

TextNet's most important technical contribution is a text-specific interface for conflict resolution and merging. While several word-by-word difference presentation systems predate TextNet, the system offers the first interactive "merge mode" that is designed for text. In simple difference representation, words can be represented as either being added or removed. In merges that take into account history, there are up to three possible texts: the text from either of the two branches being merged or the common ancestor. In my design, I've extended the difference model to include three colors for the ancestor, the current version, and the version from the branch. Changes are represented first as changed paragraphs and then as changed series of words.

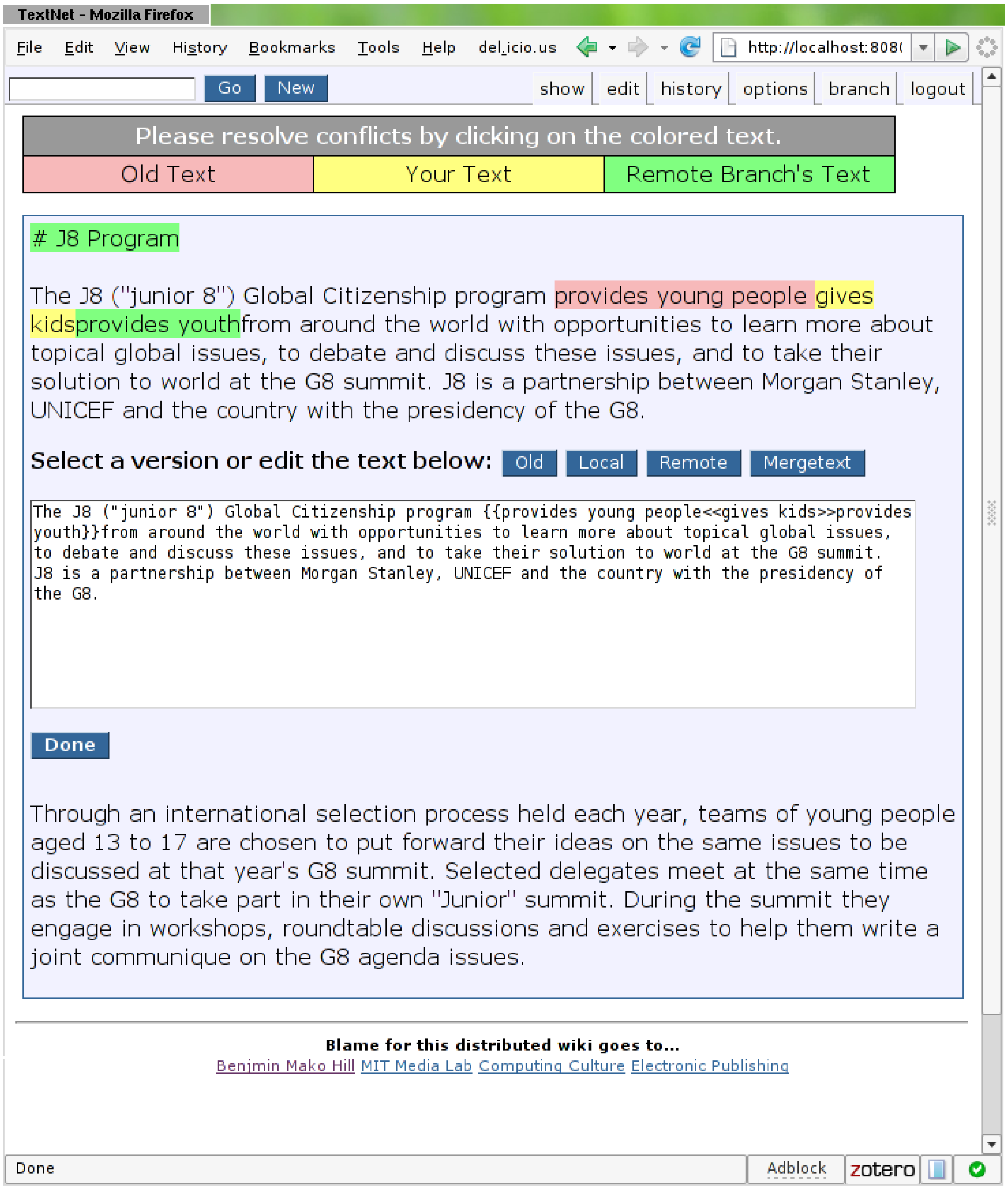
When initiating a merge, users are shown a full version of their document with conflicts highlighted using three colors and a color key at the top of the page as shown in Figure 6-7. Users are instructed to click on each conflict and to resolve them. Clicking on a paragraph brings up a conflict-resolution text box. This box contains a text field containing the the paragraph text with the conflict represented inside. The text is surrounded by double braces and different versions are separated from each other with angle brackets. As authors edit inside the text box, their changes to the paragraph are updated in real time on the page. Users are also given a simple list of actions they can take on the region. These actions include undoing any changes made and restoring the conflict text or choosing any the version of the paragraph from either branch or from the base. The interface can be seen in Figure 6-8.

In most wikis, users are asked to edit an entire page or a section at a time. TextNet employs this full-document authoring mode during normal editing but chooses the paragraph-by-paragraph model for conflict resolution. Use of a paragraph-based model is designed to help users focus on the specific conflicting regions. Users can, however, edit other paragraphs (with or without conflicts) by clicking on them at any point. Once a user has resolved a conflict or edited a paragraph to their satisfaction, they can click the “Done” button below each text box to hide the tools and edit box. When all conflicts on the page are removed, the user is invited to review and save their changes.

Before saving, TextNet presents users with a simple diff (e.g., as in Figure 6-6) showing the changes between the last saved version of the document being worked on with the draft of the the document with all the conflicts resolved and changes merged in. If the user is unsatisfied, he or she can continue and revise their changes. If satisfied, he or she can save them. Saving the changes commits a new version of the page and marks any reviewed changes from the remote branch as merged. By presenting the text specific mode for resolving conflicts upon merges, TextNet satisfies the third and most difficult requirement for CIP as laid out in the beginning of this chapter.

6.4 Wiki Interface

TextNet was originally designed as a stand-alone desktop editor. This model provided extensive control over UI and interaction design. However, based on initial feedback, the system was re-



Please resolve conflicts by clicking on the colored text.

J8 Program

The J8 ("junior 8") Global Citizenship program provides young people gives kidsprovides youthfrom around the world with opportunities to learn more about topical global issues, to debate and discuss these issues, and to take their solution to world at the G8 summit. J8 is a partnership between Morgan Stanley, UNICEF and the country with the presidency of the G8.

Select a version or edit the text below:

```
The J8 ("junior 8") Global Citizenship program {{provides young people<<gives kids>>provides
youth}}from around the world with opportunities to learn more about topical global issues,
to debate and discuss these issues, and to take their solution to world at the G8 summit.
J8 is a partnership between Morgan Stanley, UNICEF and the country with the presidency of
the G8.
```

Through an international selection process held each year, teams of young people aged 13 to 17 are chosen to put forward their ideas on the same issues to be discussed at that year's G8 summit. Selected delegates meet at the same time as the G8 to take part in their own "Junior" summit. During the summit they engage in workshops, roundtable discussions and exercises to help them write a joint communique on the G8 agenda issues.

Blame for this distributed wiki goes to...

Benjmin Mako Hill MIT Media Lab Computing Culture Electronic Publishing

Figure 6-8: Conflict resolution within a paragraph in TextNet

designed as a wiki. As a wiki, TextNet's conceptual goals remain unchanged; however, the design underwent major reevaluation and many of the most onerous technical and design challenges were associated with ensuring that the system operates effectively on the web.

TextNet's user interface is based on Samat Jain's minimalist *NullBook* skin for Mediawiki. From a functional perspective, the layout closely follows Mediawiki which is familiar to many wiki users. TextNet uses explicit wiki-style markup, which offers both advantages and disadvantages over WYSIWYG editing. WYSIWYG systems frequently have trouble graphically representing changes in markup (e.g., showing the addition or removal of bold-facing). The use of wiki text avoids the need for replicated text, supra-textual annotation, and omitted information associated with common WYSIWYG solutions to representing markup changes; every markup change *is* a change to the wiki text and is clearly visible in diffs.

Perhaps most importantly, wikis allow for editing in familiar, if not featureful, text editing interfaces through their use of simple HTML `<textarea>` boxes. Because WYSIWYG on wikis is highly wiki-specific, I chose to avoid in-browser WYSIWYG editing in order to open the possibility of use of TextNet's conflict resolution mode in conjunction with many existing wikis and dialects of wiki markup. To mitigate the drawbacks of this approach, TextNet makes extensive use of pure Javascript wiki-text rendering. As a result, changes made in TextNet text boxes can be rendered in real time on the page. Through this process, many of the benefits of a WYSIWYG are joined with the benefits of cross-platform and inter-wiki compatibility and more lightweight text-entry interfaces.

TextNet borrows heavily from other wikis in that it allows viewers to quickly and easily transition to authoring. However, while collaboration in most wikis is based around the requirement of a single version of a document or page at any given point, TextNet encourages authors to, if necessary, make their changes in a new branched page. By displaying a full view of history and offering full decision-making power to every collaborator, TextNet uses a wiki-like approach that emphasizes transparency. However, TextNet breaks the highly centralized nature of wikis by allowing parallel work in ways that traditional wikis do not. When combined with the text-specific modes of difference representation, systems for branch accounting, and the TextNet model of conflict resolution for text, TextNet offers a compelling design for the support of CIP for writing.

6.5 Technical Design

TextNet is implemented primarily in the Python programming language. Python was selected for its wide selection of relevant libraries and because of its tight integration into the One Laptop per Child XO-1 environment. Potential inclusion of TextNet in OLPC was an important long-term goal for the system. Implementation in a language other than Python would have incurred additional dependencies, which may have made deployment on OLPC less likely.

Additional code for TextNet, including most of the TextNet conflict resolution system requiring real-time feedback and interaction with the users, is written in Javascript. Javascript was selected because it is cross-platform, browser independent, does not require the installation of additional software, and is one of the programming environments available on the OLPC XO-1.

Rather than reimplementing features in extant SCM and DSCM tools, TextNet makes extensive use of *bzrlib*, the Python API interface to the Bazaar DSCM system. While there were several other compelling DSCM systems on which to base TextNet — most notably Git and DARCS — Bazaar is implemented in Python and, as a result, incurs minimal extra dependencies while maximizing integration simplicity on the XO-1. TextNet uses Bazaar as a database for differential storage and retrieval and uses bzrlib's implementation of mesh merging.

TextNet's web interaction is built using the *web.py* web framework. Built by Aaron Swartz, web.py's primary design goal is to be lightweight and flexible. Because TextNet uses a Bazaar back-end as a database, it does not use a relational database. Without the need for a database handler or an object relational mapper, web.py served as an ideal base for simple request and session handling and web server integration.

For presentation and user interface programming, TextNet uses *Cheetah* templates. Implemented in Python, Cheetah templates creates a strong barrier between presentation logic and programming logic which will make integration of TextNet into other wiki environments and back-ends easier in the future.

For wiki text rendering, TextNet targeted OLPC's *Crossmark* specification. Unfortunately, no mature or tested implementation of Crossmark exists to date. Instead, TextNet uses the *Markdown* markup language — Crossmark's most direct inspiration. Markdown is a simple wiki language that makes the creation of "invalid" markup difficult. Because it is widely used, there exist several implementations including libraries in Python and Javascript that TextNet was able to use.

Finally, TextNet employed the *Showdown* library to display real-time rendering of changed Markdown text. Showdown is a simple Javascript Markdown-to-HTML converter implemented in Javascript. Showdown is reasonably fast, cross platform, and runs without modification on the OLPC Javascript environment. A simple version using *Instaview* supporting Mediawiki markup was also created but was not evaluated.

6.6 Design Timeline

TextNet's initial target audience were editors in the Citizendium project. At the time, Citizendium was planned as a sustained parallel "fork" of Wikipedia and was an ideal test-case for support and evaluation of tools designed to support CIP for writing. In January 2007, I designed a first version of a TextNet interface for this target. This version was a stand-alone merge-tool using the *PyGTK* desktop library. The design was built to interact with Mediawiki but was not, itself, a wiki. This technical design introduced a high degree of control of the user interface and and interaction modes. The system included a module based on the Mediawiki *PyBotFramework* that could extract and duplicate articles from Mediawiki instances and could, through this process, generate DSCM branches from existing Medaiwiki pages including both Citizendium and Wikipedia.

In March, the Citizendium team made a decision to deviate from their previous plan and to write their encyclopedia from scratch instead of as a fork of Wikipedia. Without a base in Wikipedia, Citizendium's use for CIP support was reduced dramatically. At the same time, OLPC was struggling with issues connected to merging offline and parallel work in the context of wiki

Date	Version	Milestone
January 15	0.0.1	non-web interface for use with Citizendium
March 1	0.0.2	redesign as a stand-alone wiki appropriate for OLPC
May 7	0.1	working version with demonstrations at ML sponsors week
May 15	0.2	iterations based on feedback
June 1	0.3	OLPC version of TextNet (MikMik) created for J8 Summit
July 15	0.4	final version of tool for final evaluation

Table 6.1: Design-timeline table

authoring. In response, I redesigned TextNet as a Python wiki usable both on OLPC and as a stand-alone tool.

I demonstrated a working version of TextNet in the first week of May for the MIT Media Lab "sponsors week." Subsequently, I iterated on the design based on informal feedback during demonstrations and on discussions with individuals at OLPC.

OLPC's Director of Content SJ Klein invited me to a meeting at UNICEF in mid-May where I demonstrated TextNet to a team at UNICEF's Division of Communication that, among other projects, works to support collaborative writing among highly disconnected groups of young people. The group was was interested in using OLPC XO-1's to support collaborative writing at the J8 summit. Working with interns at OLPC, I prepared a OLPC "activity" (i.e., a bundled application that can easily be installed on the XO-1) of TextNet called *MikMik* over the first of June. While, for technical reasons, the tools was not widely used at the J8 meeting, UNICEF staff members worked with me to help evaluate TextNet based in part on their experiences at the summit.

Chapter 7

Evaluation

TextNet was evaluated in three ways. First, a simple engineering analysis evaluated the systems ability to fulfill a set of features goals stated in the initial proposal for the system. Second, an in-depth review of the system for usability with a usability expert who was actively engaged in the one of the system's target communities evaluated TextNet's ability to provide this functionality in ways that would be accessible and discoverable. Finally, and most importantly, a series of user tests evaluated the software's ability to take advantage of features central to CIP. While results of the initial evaluation are mixed, they show potential for the system in several important areas and provide a clear set of goals for future development and further work.

Much of TextNet's evaluation involved work with a team within UNICEF's Division of Communication. Through a series of projects, including the J8 summit described in depth in Section 2.1, the department is heavily involved in collaborative writing with young adults — particularly on wikis. Due to UNICEF's extensive work with communities with intermittent Internet connectivity, the group is interested in a system like TextNet to help support offline collaboration. Because of the J8 and a series of other projects, the group is also interested in pursuing other forms of CIP with young adults.

I worked with UNICEF before, during, and after the J8 conference. In addition to the evaluation described here, earlier iterations of the system were designed with informal input from UNICEF staff members and several versions were tested informally and iterated on based on early feedback. Evaluation of *TextNet 0.4* (see Table 6.6) with UNICEF centered on their extensive experience using wikis and other collaborative writing tools with children and with their hands-on experience acting as technical facilitators during the J8 summit.

7.1 Engineering Evaluation

The most simple method of evaluation evaluates TextNet's ability to satisfy the initial feature goals. This list of feature goals included (1) functionality to discover and to track the status of branches; (2) functionality to provide history sensitive merging between wikis running on separate servers; (3) an ability to provide a wiki-based in-line or word-based method of displaying changes; and (4) features to represent and resolve conflicts using a word-based interface. The current version of *TextNet 0.4* released July 15 includes tested functionality satisfying all four goals.

7.2 Usability Review

The system was evaluated for usability in a 45-minute session with a UNICEF program manager and usability expert. This expert has ten years of experience in UNICEF and his work focuses on user interface evaluation, improvement, and design. He has experience with wikis through the the *MoinMoin* wiki software used by UNICEF during the J8 summit, helped provide usability feedback on that system and has made several edits on Wikipedia, but has no other wiki experience. His only other experience with collaborative writing systems is what he describes as extensive experience with Microsoft Word's track changes features described in Section 5.4. The expert used the system for 15 minutes, in parallel with me, tested all the core features of the system, then did a 30-minute-long walk-through and evaluation of the system.

The usability expert was most supportive of the the in-line display of word-by-word differences, which he felt were a step forward over other wikis. He was supportive but less enthusiastic about the paragraph-by-paragraph conflict resolution mode. He felt that the description of "Old Text," "Your Text," and "Remote Branch's Text" were unclear in the merge interface and suggested that users would have trouble determining where changes were coming from. While he felt that the conflict resolution mode was reasonably simple, he also believed that important concepts about branch relationships essential to the system's effectiveness would be undiscoverable for many users.

Additionally, he expressed concern about the use of color in the merge interface which might render the system inaccessible to color-blind users and suggested that using differences in font style and weight in addition to judicious choice of variation in color detectable by colorblind users would mitigate this issue. He suggested that the complexity of the system would become prohibitively complex with networks of active collaborators larger than three or four collaborators. The largest user tests involved only three individuals, so his hypothesis remains untested.

Finally, the expert was very critical about the lack of a WYSIWYG interface which, at the J8 summit and and other work, provided a significant barrier to use by young people. A review of other wiki usability studies suggested that this concern — while an important usability barrier — is an issue common to and inherited from other wikis [18, 60].

7.3 User Tests

The system was tested with eight people at the UNICEF Division of Communication at UNICEF headquarters in New York City. Each user had been involved in support of collaborative writing using wikis during the J8 summit. The users had varying degrees of technical experience and different roles at UNICEF. Only one person had actively used wikis before the J8. Five were familiar with Word's track changes functionality and three users had experience with Google's collaborative document editor. Most users claimed that they were comfortable with wikis but only one use — a programmer — expressed that he found them "straightforward," and, "very easy to use."

Testers used the system in two groups of two. The four remaining users used it individually but merged their changes with that of other UNICEF members who had used the system previously.

Before evaluation, TextNet was briefly described to each user without reference to any particular use cases. Four users suggested potential uses for the system and three mentioned the potential for the coordination of off-line work. One group of two recognized the potential for maintenance of diverged document comparing it favorably to Google docs and complaining that Google docs made it extremely difficult to have multiple copies of a single document suggesting that it was, "more like *overwriting* than writing."

Each user was given a task of editing a short document about the J8 summit based on text from the Wikipedia article on the subject [7]. Several obvious mistakes were inserted into the text before testing in ways that made the "fix" non-obvious. Notes were taken during each test and two tests were recorded with video. Users were given the following tasks:

1. Edit an existing version of the article removing any obvious errors to the document and adding or changing as much or as little as desired;
2. Merge changes from a (seeded) branch that had also fixed the obvious errors. This involved resolving at least two conflicts in each case;
3. Improve the article in a substantive way. This was intentionally left open-ended and was executed concurrently with another editor given the same task but working in a different branch — a role that I filled myself;
4. Merge changes from the branch under concurrent editing and resolve any resulting conflicts in the process;
5. Create a new branch based off the current branch.

Each user completed at least the first four tasks although most had at least some assistance. Users took an average of 15 minutes to complete the tasks and spent an additional 10–15 minutes discussing the system immediately following their use of TextNet. The second merge task was, in almost every case, significantly faster than the first (on average, less than 40% of the time). Every user was able to resolve the conflicts although two unintentionally lost at least some of their their own changes.

Three users used only the buttons in the merge interface before being prompted and three preferred to edit the wiki text; two used a mixture of both. All users who edited in the text boxes understood the wiki text representation of conflicts and editing involved to resolve them without explanation. Users were, however, much less clear about the use of the merge-interface buttons and were, echoing the evaluation described in Section 7.2, initially unclear as to what "Old Text," "Your Text," and "Remote Branch's Text," referred to. When users were shown a drawing of a simple branch diagram representing their merge operation, they showed signs of immediate understanding. One user admitted that she had not, "really understood what [she] was doing," before seeing the drawing.

Testing revealed several important usability issues. The use of similar highlight colors and backgrounds in the merge mode screen and the change review summary led both video-taped users to attempt to click on the view text in the change summary screen. Users also had difficulty finding the TextNet URLs and creating branches of existing documents which cannot currently be done from the document to be branched from as several users assumed. Users understand, but did not comment on, the word-by-word difference representation system.

Account management on the *MoinMoin* J8 wiki led to major headaches during the J8 Summit. Photo uploading continues to be one of the most popular features among the youth UNICEF works primarily with. Users suggested that TextNet's requirement to create accounts and the inability of TextNet to upload photos would prove problematic during the J8 Summit or at a similar event.

Two user expressed a desire for a WYSIWYG interface although no user made active use of markup during their editing. Five users described what they felt was potential for the system in the J8 or in future events but two suggested that they would not want to replace the current *MoinMoin* system and instead might want to use TextNet or similar tool primarily for users working offline or to allow UNICEF staff members to merge changes for the youth. One suggested that the system, as is, might be too conceptually complex for young people or novice users.

7.4 Lessons Learned

Before evaluation, I hypothesized that usability issues and difficulties in using TextNet's conflict-resolution system would present the most pressing problems for users. In fact, evaluation both with the usability expert and in user tests demonstrated that the most difficult barriers to effective use of TextNet were rooted in the underlying complexity of the system and users' unfamiliarity with the concepts and processes at play rather than with the interface. Users were able to review and resolve changes but did not, however, always understand where the changes were coming from. A future version of TextNet may be able to address this by supporting explicit graphical branch visualizations that shows users the relationships of branches involved in a merge operation.

These issues are perhaps indicative of a deeper problem: users are able to learn *how* to use TextNet but, in practice, do not always know *when* to use it or *why* they should — even when they understand and are excited by the concepts and purpose of the system. Cooperation in parallel is a mode of cooperative writing that is foreign enough from normal modes of group writing that any method of support may faces similar challenges. TextNet seems to do an adequate, if imperfect, job of supporting CIP. However, it seems incomplete for many users without doing a better job of conveying what CIP is and what it looks like from within the system.

Chapter 8

Conclusion

I am an active software developer and a passionate advocate of free software. It is through my work in FOSS communities, and in Ubuntu in particular, that I first experienced and came to understand the power of the development methodology that I've called cooperation in parallel. It was my experience with CIP in free software communities that motivated me to explore the facilitation of similar modes of collaboration in my *other* passionate creative pursuit: writing.

This thesis is an exposition and an exploration of this new, powerful, mode of work. It attempts to defined and begin to understand CIP so that it can be harnessed, where appropriate and where possible, toward more and effective collaboration. It analyzes CIP theoretically by looking to the CSCW and CSCWr communities and to FOSS developers using SCM and DSCM tools. It explores the differences between writing and writing code in a way that tries to inform the analogies between software and writing at the heart of this process and to add a social dimension to the description of CIP for documents. It builds on important related technologies in order to offer an informed design for a system that might support cooperative, parallel writing. It evaluates that system in its ability to support the core technical principles at the heart of CIP.

As an argument for a new type of collaborative work, the motivation for this paper — if not the core theses themselves — is inherently speculative. While CIP is useful for software developers today, the real potential for CIP for writing is those places we have not yet imagined. CIP for software is like a vocabulary or a set of terms that can be used to communicate a new set of concepts; it is difficult to explain what a vocabulary lets one communicate to someone without that vocabulary. In this early period, CIP for writing is in an analogous situation.

TextNet offers a solid, if rough, first attempt at supporting CIP in writing. DSCM tools are being written and revised by FOSS communities to support evolving methods in FOSS communities. TextNet and similar tools will also need to be developed through a long iterative process alongside collaborative writing communities. Before succeeding, these technologies and communities will also need to grow and develop together — in parallel.

Bibliography

[1] Bernstein v. US Department of State, 1996. 922 F. Supp. 1426.

[2] Bernstein v. US Department of State, 1996. 945 F. Supp. 1279.

[3] Bernstein v. US Department of State, 1997. 974 F. Supp. 1288.

[4] Alphabetical list of programming languages. Wikipedia, July 2007. Available: http://en.wikipedia.org/wiki/Alphabetical_list_of_programming_languages.

[5] Article validation proposals. Meta Wikimedia, Aug. 2007. Available: http://meta.wikimedia.org/wiki/Article_validation_proposals.

[6] G8. Wikipedia, July 2007. Available: http://en.wikipedia.org/wiki/G8.

[7] J8. Wikipedia, July 2007. Available: http://en.wikipedia.org/wiki/J8.

[8] List of basic dialects. Wikipedia, July 2007. Available: http://en.wikipedia.org/wiki/List_of_BASIC_dialects.

[9] Polyglot (computing). Wikipedia, Aug. 2007. Available: http://en.wikipedia.org/wiki/Polyglot_(computing).

[10] Revision control. Wikipedia, July 2007. Available: http://en.wikipedia.org/wiki/Revision_control#Distributed_revision_control.

[11] Baldwin, C. Y., and Clark, K. B. *Design Rules, Vol. 1: The Power of Modularity*. The MIT Press, Mar. 2000.

[12] Beck, E. E. *A survey of experiences in collaborative writing*. Springer-Verlag, Germany, 1993, pp. 9–28.

[13] Black, M. J. *The art of code*. PhD Dissertation, University of Pennsylvania, 2002.

[14] Bly, S. A., Harrison, S. R., and Irwin, S. *Media spaces: bringing people together in a video, audio, and computing environment*, vol. 36. ACM Press, 1993.

[15] Bond, G. W. *Software as art*, vol. 48. ACM Press, 2005.

[16] Bruns, A. Wikinews: The next generation of alternative online news? *Proceedings Association of Internet Researchers Conference, Chicago* (2005).

[17] Bryant, S. L., Forte, A., and Bruckman, A. Becoming wikipedian: transformation of participation in a collaborative online encyclopedia. In *GROUP '05: Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work* (New York, NY, USA, 2005), ACM Press, pp. 1–10.

[18] Desilets, A., Paquet, S., and Vinson, N. G. Are wikis usable? *The 2005 International Symposium on Wikis. San Diego, California, USA. October* (2005), 17–18.

[19] Dillenbourg, P., Baker, M., Blaye, A., and O'Malley, C. The evolution of research on collaborative learning. *Learning in Humans and Machine: Towards an interdisciplinary learning science* (1996), 189–211.

[20] Dix, A., and Miles, V. C. Version control for asynchronous group work. YCS 181, Department of Computer Science, University of York, (Poster presentation HCI'92: People and Computers VII), 1992.

[21] Dix, A., Rodden, T., and Sommerville, I. Modeling versions in collaborative work. In *IEE Proceedings in Software Engineering* (1997).

[22] Dourish, P., and Bellotti, V. Awareness and coordination in shared workspaces. In *CSCW '92: Proceedings of the 1992 ACM conference on Computer-supported cooperative work* (New York, NY, USA, 1992), ACM Press, pp. 107–114.

[23] Dowler, A. Integration guide for Microsoft Office 2003 and Windows SharePoint Services. Microsoft TechNet, June 2003. Available: http://www.microsoft.com/technet/windowsserver/sharepoint/v2/spoffint.mspx.

[24] Ellis, C. A., Gibbs, S. J., and Rein, G. L. Groupware: Some issues and experience. *Communication of the ACM 34* (1991), 38–58.

[25] Esolang. Esoteric programming languages. Esolangs Wiki, July 2007. Available: http://esoteric.voxelperfect.net/wiki/Esoteric_programming_languages.

[26] Fogel, K. F. *Open Source Development with CVS*. Coriolis Group Books, 1999.

[27] Fogel, K. F. *Producing Open Source Software: How to Run a Successful Free Software Project*. O'Reilly, 2005.

[28] Foundation, F. S. GNU coding standards. Free Software Foundation, June 2007. Available: http://www.gnu.org/prep/standards/html_node/index.html.

[29] Gelernter, D. H. *Machine Beauty: Elegance and the Heart of Computing (Repr ed)*. Basic Books, 1999.

[30] Gonzalez-Barahona, J. M., Perez, M. A. O., de las Heras Quiros, P., Gonzalez, J. C., and Olivera, V. M. Counting potatoes: the size of Debian 2.2. *Upgrade Magazine 2* (2001), 60?66.

[31] Grimmelmann, J. Incantations, legal and computational. PrawfsBlawg, May 2007. Available: http://prawfsblawg.blogs.com/prawfsblawg/2007/05/incantations_le_2.html#more.

[32] Hassine, T. The dynamics of NPOV disputes. *Proceedings of Wikimania 2005: The First International Wikimedia Conference* (2005).

[33] Henricson, M., and Nyquist, E. *Industrial Strength C++: Rules and Recommendations*. Prentice Hall, Sept. 1996.

[34] Hill, B. M. Literary collaboration and control. Available: http://mako.cc/projects/collablit/, May 2003.

[35] Hill, B. M. To fork or not to fork: Lessons from ubuntu and debian. In *Processings of LinuxTag* (Karlsruhe, Germany, July 2005).

[36] Hill, W. C., Hollan, J. D., Wroblewski, D., and McCandless, T. Edit wear and read wear. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 1992), ACM Press, pp. 3–9.

[37] Husted, K., and Micailova, S. Diagnosing and fighting knowledge-sharing hostility. *Organizational Dynamics 31* (Aug. 2002), 60–73.

[38] Jones, S. *MILO: A Computer-based tool for (co-)Authoring structered documents*. Springer-Verlag, Germany, 1993, pp. 185–202.

[39] Kastrup, D. Revisiting WYSIWYG paradigms for authoring LaTeX. *Proceedings of the 2002 Annual Meeting, TUGboat 23* (2002).

[40] Kirby, A., Rayson, P., Rodden, T., Sommerville, I., and Dix, A. Versioning the web. In *7th International Workshop on Software Configuration Management* (Boston, MA, 1997), R. Conradi, Ed., pp. 163–173.

[41] Lakhani, K. R., and McAfee, A. P. Harvard Business School case: Wikipedia (A), 2007. Available: http://courseware.hbs.edu/public/cases/wikipedia/.

[42] Lakhani, K. R., and Wolf, R. G. Why hackers do what they do: Understanding motivation and effort in free/open source software projects. *Perspectives on Free and Open Source Software* (2005).

[43] Lessig, L. *Code and Other Laws of Cyberspace*, 2nd ed. Basic Books, June 2000.

[44] Leuf, B., and Cunningham, W. *The Wiki way: quick collaboration on the Web*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2001.

[45] Leyton, R. Version control: A misunderstood technology. Leyton.org, June 2005. Available: http://www.leyton.org/diary/2005/06/06/version-control-a-misunderstood-technology/.

[46] Lippman, A., Bender, W., Soloman, G., and Saito, M. Color word processing. *Computer Graphics and Applications, IEEE 5* (1985), 41–46.

[47] Lipshaw, J. More on similarities between programming language and the language of law. Legal Profession Blog, May 2007. Available: http://lawprofessors.typepad.com/legal_profession/2007/05/more_on_similar.html.

[48] Lowry, P. B., Curtis, A., and Lowry, M. R. Building a taxonomy and nomenclature of collaborative writing to improve interdisciplinary research and practice. *Journal of Business Communication 41* (2004), 66.

[49] Mateas, M., and Montfort, N. A box, darkly: Obfuscation, weird languages, and code aesthetics. *Proceedings of digital arts and culture* (2005).

[50] Mingst, K. Eight, group of, July 2007. Available: http://www.britannica.com/eb/article-9403959.

[51] Moeller, E., and Hill, B. M. Definition of Free Cultural Works, 2007. Available: http://www.freedomdefined.org/.

[52] Nelson, T. Project xanadu history, 2001. Available: http://xanadu.com/HISTORY/.

[53] Nelson, T. H. *Literary Machines*, 3rd ed. Self Published, Swarthmore, PA, 1981.

[54] Park, R. E. *Software Size Measurement: A Framework for Counting Source Statements*. Carnegie Mellon University, Software Engineering Institute, 1992.

[55] Perens, B. The open source definition. *Open Sources: Voices from the Open Source Revolution* (1999), 171–188.

[56] Petkovsek, M., Wilf, H., and Zeilberger, D. $A=B$. A.K. Peters, Ltd., 1996.

[57] Posner, I. R., and Baecker, R. M. How people write together. *Proceedings of the Twenty-fifth Annual Hawaii International Conference on System Sciences 127* (1992), 138.

[58] Raymond, E. S. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly and Associates, Sebastopol, CA, 1999.

[59] Reagle, J. A case of mutual aid: Wikipedia, politeness, and perspective taking, 2004. Available: http://reagle.org/joseph/2004/agree/wikip-agree.html.

[60] Reitmayr, E. Usability test results: Editing information in the german wikipedia, Feb. 2006. Available: http://de.wikipedia.org/wiki/Wikipedia:WikiProjekt_Usability/Test_Februar_2006.

[61] Rosenberg, J. Some misconceptions about lines of code. *Proceedings of the 4th International Software Metrics Symposium (Metrics ?97)* (1997), 137–142.

[62] Schade, O., Scheithauer, G., and Scheler, S. One program in 1111 variations, 2007. Available: http://www.99-bottles-of-beer.net/.

[63] Schmidt, K. The problem with awareness: Introductory remarks on awareness in CSCW. *Computer Supported Cooperative Work (CSCW) 11* (2002), 285–298.

[64] Searle, J. R. *Speech Acts: an essay in the philosophy of language*. Cambridge University Press, 1980.

[65] Sharples, M., Goodlet, J. S., Beck, E. E., Wood, C. C., Easterbrook, S. M., and Plowman, L. *Research issues in the study of computer supported collaborative writing*. Springer-Verlag, Germany, 1993, pp. 9–28.

[66] Skaf-Molli, H., Ignat, C., Rahhal, C., and Molli, P. New work modes for collaborative writing. In *EISWT-07* (2007).

[67] Stallman, R. M. *Free Software, Free Society: Selected Essays of Richard M. Stallman*. Free Software Foundation, Oct. 2002.

[68] Tatar, D. G., Foster, G., and Bobrow, D. G. *Design for conversation: lessons from Cognoter*. Academic Press Ltd., 1991.

[69] Tichy, W. F. *Design, implementation, and evaluation of a Revision Control System*. IEEE Computer Society Press, Tokyo, Japan, 1982.

[70] Torvalds, L. Kernel scm saga.., Apr. 2005. Available: http://kerneltrap.org/node/4968.

[71] UNICEF. About j8. Junior 8, 2007. Available: http://www.j8summit.com/summit/showpage.php?id=362.

[72] van Rossum, G. *Python Reference Manual*. Centrum voor Wiskunde en Informatica, 1995.

[73] Vaughan-Nichols, S. J. Details emerge about Debian linux plan, July 2005. Available: http://www.eweek.com/article2/0,1895,1836184,00.asp.

[74] Vaughan-Nichols, S. J. Linspire, Canonical, Freespire, Ubuntu join forces. *Desktop Linux* (Feb. 2007). Available: http://www.desktoplinux.com/news/NS7103672739.html.

[75] Viegas, F. B., Wattenberg, M., and Dave, K. Studying cooperation and conflict between authors with history flow visualizations. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 2004), ACM Press, pp. 575–582.

[76] Voss, J. Measuring wikipedia, 2005. Available: http://eprints.rclis.org/archive/00003610/.

[77] Wales, J. Open plenary, Aug. 2006. Available: http://wikimania2006.wikimedia.org/wiki/Opening_Plenary_%28transcript%29.

[78] Wales, J., and President, W. F. Wikipedia sociographics. Talk at the 21C3-Conference, 2004.

[79] Walker, L. On local sites, everyone's a journalist. *The Washington Post* (Dec. 2004), E01.

[80] Wei, C., Maust, B., Barrick, J., Cuddihy, E., and Spyridakis, J. H. Wikis for supporting distributed collaborative writing. In *The proceedings of the annual conference of the Society for Technical Communication 52nd Annual Conference* (Arlington, VA, 2005), pp. 8–11.

[81] Wheeler, D. A. Comments on Open Source Software / Free Software (OSS/FS) Software Configuration Management (SCM) systems, May 2005. Available: http://www.dwheeler.com/essays/scm.html.

[82] Wheeler, D. A. The most important software innovations, June 2007. Available: http://www.dwheeler.com/innovation/innovation.html.

[83] Williams, S. *Free as in Freedom: Richard Stallman's Crusade for Free Software*, 1 ed. O'Reilly Media, Inc., Mar. 2002.

[84] Yeomans, M. The birth of Wikinews. Citizen's Kane, 2005. Available: http://citizenskane.blogspot.com/.

Acknowledgments

Thanks to my thesis adviser Walter Bender who was willing to work with me on this thesis despite having been on leave from the lab for the complete duration of the project. He took time out of an unimaginatively busy schedule to help me. As president of OLPC, Walter *really did* have better things to do than work with me on a only tangentially related Masters thesis. His help and direction was always helpful and my work is better as a result. I will treat his work with me as an investment that I hope to repay to him and the community he took time away from with some of the skills I've learned in this process.

Thanks also to Chris Csikszentmihályi for advice, time, effort, and arguments. In Chris's group, I found the best home a lab "orphan" could hope for. At times, my work pushed the limits of what Chris was comfortable with — a reversal of roles for Chris perhaps — but he challenged himself to see the benefits and relevance of my work that was both coming from and serving fields he was not initially familiar with. I can't thank him enough for his effort, energy, and flexibility.

Thank you to Biella Coleman for working with me on the chapter on writing and writing code. Biella helped me add a layer of reflectiveness — and a type of methodology — that most Media Lab thesis go without. I believe that other theses are poorer without someone like Biella to help direct them.

Thanks as well go to the Computing Culture group including Adam Whiton, Alyssa Wright, Annina Rüst, Kelly Dobson, and Noah Vawter, in addition to CompCult UROPs Courtland Allen, John Dong and Justin Sharps. This project has interrupted and affected our work and friendships but you've provided a fantastic environment for learning and support. Also thanks to my Electronic Publishing colleagues Dustin Smith, Bo Morgan, Ian Eslick and Hyemin Chung who provided a welcoming home for me during my first year at the lab.

It seems cliché to thank your partner for putting up you with during the process of a long research or writing project but I can't think of an acknowledgment that's more deserved. The many late nights, glazed looks, and panicked moments during the process of creating this document were difficult for me but I always had a future feeling of accomplishment to motivate me. Mika had no such reward but patiently and supportively endured the process with me. I can't thank her enough.

Thanks as well to my publisher Deborah Williams-Cauley and my friend Joshua Gay for generously taking time out to proofread my thesis. Your work is invisible but it's not unappreciated.

Final thanks go to my friends and cohorts from the Boston and Cambridge free software and free culture communities and in particular, the group of people on `#yukidoke` and `#acetarium`. You're both the inspiration for this work and the community I hope takes advantage of it to achieve even greater things.