TraceBack

MOO, Open Source, and the Humanities

Jan Rune Holmevik

Dr. Art. Thesis
Dept. of Humanistic Informatics
Faculty of Arts, University of Bergen
Spring 2004

ISBN 82-497-0189-5

Contents

Acknowledgements

Every journey must come to an end and so must the journey that is a dissertation. In one way or another I have been involved with the technologies, methodologies and practices that are the subject of this study for the last 10 years. I still have vivid memories from my first encounters with Linux, MOO, Email, the World Wide Web and many of the other technologies that are covered within these pages. I have lived with them and through them for so long that they are part of who I am. This study, therefore, is also a personal story about my own coming of age as an observer, scholar, participant, and creator of technologies that influence our lives.

Journey's end is an appropriate time to contemplate what you have done and experienced on a long voyage. It is also a time to acknowledge those who have helped and supported you along the way. I have been very fortunate to have had the support of many great people who I consider not only to be my sharpest peers, but also close and personal friends. First and foremost I wish to extend my sincere gratitude to my dissertation supervisor, Professor Espen Aarseth. He believed in me right from the start, and over the years he has been a true source of support, inspiration and gentle encouragement. The opportunity he gave me to chair the Digital Arts & Culture Conference in 2000 was a gesture for which I am deeply and utterly grateful.

A substantial part of this project has centered on the development of an Open Source software package called enCore. In this work I have benefited from a number of contributions of code from MOO programmers all across the world. Some of them have contributed large parts, some small, but I am truly grateful for everything they have done. I would therefore like to take this opportunity to thank them each individually. In no particular order, their names are: Pavel Curtis, Cynthia Haynes, Sindre Sørensen, Ken Schweller, Mark Blanchard, Jorge Barrios, Amy Bruckman, Matthew Campbell, John Towell, Gustavo Glusman, Craig Leikis, Juli Burk, Michael Thompson, Rui Miguel Barbosa Pinto, Andrew Wilson, Ken Fox, Matthew Allen Beermann, Jason Nolan, Noel Davis, Stephen Ashley, Hervé Collin, Emma Jane Hogbin, Claudijo Borovic, Alexandre Borgia, Scott Carmichael, Michel Lavondes, and Daniel Jung.

Thanks are also due to all those who have used the enCore system to create new educational MOOs over the years. There are far too many to name, but I want you all to know that I have taken immense pride in the way you have helped further the use of MOO technology in education. Your success is my reward.

Just as the dissertation itself has been a journey in virtual worlds, so have I journeyed in the real world while creating it. I would therefore like to pay tribute to the places that have inspired my work. I wrote the prospectus for the dissertation in a small hotel room in Honolulu, Hawaii back in the summer of 1997. As I write these final words I find myself back at my family farm overlooking the Geirangerfjord on the west coast of Norway. In the interim my work has taken me to exotic and far-flung places such as Ghost Ranch in the high desert of New Mexico, where I wrote most of the code for enCore version 3.0, and Cape Town, South Africa, where I wrote the draft of what has now become

chapter six. From my office at the University of Bergen, Norway to my balmy front porch in Fort Worth, Texas, the places I have worked have inspired my writing in ways that are hard to quantify.

The significance of friendship can also be hard to quantify, but there is no doubt as to its value. It is important for me, therefore, to include my good friend, Dr. John F. Barber, in these acknowledgements. Musing over fine Belgian Ales on a shady front porch on sunny afternoons, or cruising together along the twisty back roads of Texas on a couple of well-oiled Harleys has provided much needed diversion and intellectual relief. Thanks for being there, buddy.

This project was made possible through a 3-year doctoral scholarship from the Norwegian Research Council's program Social and Cultural Premises for Information and Communication Technology (SKIKT). The Department of Humanistic Informatics at the University of Bergen provided the funding for a 4th year and also gave me the opportunity to teach in the department's undergraduate program. My heartfelt thanks go to professors Roald Skarsten and Dag Elgesem as well as my colleagues and fellow graduate students: Dr. Hilde Corneliussen, Dr. Torill Mortensen, Dr. Jill Walker, and Carsten Jopp. Finally, I'd like to acknowledge the generous support that I have received from the School of Arts and Humanities of the University of Texas at Dallas (UTD). I am especially grateful to Dean Dennis Kratz for inviting me to spend the 2000-2002 academic years as a visiting scholar at UTD. The School of Arts and Humanities has also generously hosted Lingua MOO from 1995 to the present as well as the enCore web site, email listserv, and FTP archive.

They say that one should save the best for last, and so I have done. This project, the practical programming and development aspects of it as well as the writing of the dissertation itself, would not have been possible without the

inspiration, undying support and generous assistance from my dear wife, Cynthia Haynes. She has not only been my soulmate and best friend over all these years, she has also been a true partner and colleague. She has always inspired me to do my very best and to strive for excellence in everything I do. I dedicate this work to you, Cynthia.

Geiranger, June 2003

Jan Rune Holmevik

About the enCore Software Distribution

The enCore software distribution is the technical development portion of my dissertation. It includes the latest version the complete enCore system along with all the updates that I have released since 1997. For easy reference, I have extracted the source code I wrote for the Xpress client and GUI layer. All total, the Xpress specific code is about 10.000 lines or approximately 250 pages in length. These source code files are located in the directory */scr* inside the distribution. The remainder of the enCore specific code can be viewed inside any enCore-based MOO using the standard *@dump* or *@list* MOO commands. The complete enCore software distribution can be found at the following online location:

http://lingua.utdallas.edu/encore/

For instructions on how to set up an enCore MOO please see the file "installation.html" inside the distribution itself.

Please refer to the enCore web site at http://lingua.utdallas.edu/encore/ for any up-to-date information that may have been released since this dissertation was submitted (June, 2003).

TraceBack

TraceBack

1: to map information to its point of origin in order to learn more about phenomena under investigation.

2: debugging tool for MOO programmers. A stack trace that identifies the exact point at which a run time error has occurred. It includes all verb (method) calls leading up to the verb where the error is thrown.

3: footprint, or imprint, marking the track where a new path is being cleared.

1

Introduction

Through the endless stream of 0s and 1s you can make out the features of a guy about 6'3. He's got dark curly hair, bright blue eyes, and he is wearing black Levis, black boots, and a white t-shirt. He is awake and looks alert—MediaMOO, 1994.

I took the Internet to purple-crayon.media.mit.edu 8888. It was the spring of 1994 and I was a graduate student in the history of technology at the Norwegian University of Science and Technology. From my little loft office overlooking the Frogner Park on the west side of Oslo, Norway, I surfed across virtual landscapes I never dreamed existed and along the way I met real people who influenced my life and career in ways I could never have imagined. Purple-crayon was a computer at the Massachusetts Institute of Technology (MIT) and the home of a new and exciting academic community, a digital space called MediaMOO. For me it became the first stop on an intellectual journey that has lasted almost 10 years. This dissertation is my logbook from that journey. It is partly a story about the creation of some of our time's most prominent information and communication technologies (ICT), partly a record of my own personal struggle with questions and issues pertaining to ICT in teaching and research which

resulted in new technological solutions to the pedagogical problems of teaching with one such technology. At the core, however, this dissertation is also an investigation into how we, as scholars and educators in the humanities (and specifically the field of Humanistic Informatics), may approach some of the challenges that these modern information and communication technologies raise.

The fundamental question that became the point of departure for this work stemmed from a very simple observation. Since the creation of the personal computer in the 1970s and the proliferation of the Internet in the 1980s and 90s, information and communication technologies have become more and more integrated in our daily lives and their impact on society and culture at large is now so vast that it is often transparent and hard to spot if we do not know what came before. There are no signs that this development is going to abate or become less important in the future. To the contrary, new information technologies emerge at an alarming pace and overwhelm us every day with promises of a better future. Historically, the research and development of ICT have been the domain of computer scientists, software and hardware engineers and other computer professionals, while we as scholars and educators in the humanities have mostly been passive, and in some cases reluctant, adopters and users of technologies that are being handed to us. For me, this observation begged a very simple question: How can we as scholars and educators in the humanities become more actively involved in the conceptualization and creation of our own technological future? In response to this question I identified three essential challenges that I believe must be addressed.

The first challenge centers on our use of ICT. In order to understand the inherent possibilities and problems with current technologies, we have to actually use them in our teaching and research. Unless we have a good and

thorough understanding of how a particular technology works or doesn't work, we can be in no position to help improve it or come up with better alternatives. Fortunately, this challenge is no longer as critical as it once was. Over the past few decades, what we may call special interest groups within most, if not all, academic fields have to varying degrees moved to incorporate ICT in their teaching and research practices. Computers have become a common sight in many humanities classrooms, and the resistance to technology that prevailed just ten to fifteen years ago is now fading as a new generation of educators who grew up with the personal computer and the Internet are moving into faculty positions everywhere.

The second challenge has to do with our practical and theoretical understanding of modern information and communication technologies, their creation, use, and effects on society and culture. To be sure, a good number of scholars in the humanities have also wrestled with this challenge for a long time already. In the field of history, for example, scholars have for many years been interested in how new information technologies can be used in pursuit of historical research, analysis, and presentation. In the English, Rhetoric and Composition fields, scholars have formed the sub-field of Computers and Writing to study the use of information technology in relation to teaching and research. Computer-assisted language learning has long traditions in many foreign language departments. Other Humanities disciplines have taken similar initiatives in regard to the specific challenges they see for their respective fields. On the theoretical end of the spectrum, scholars in fields such as History of Science and Technology and Studies of Technology and Society (STS) have done extensive research on the creation of ICT and the effects on individuals, society

and culture. In literature, scholars have studied the emergence of new literary genres made possible by new digital media. The list could go on.

All these initiatives are good and necessary, but to my mind they fall short of meeting today's more complex challenge. This is in part due to the fact that some of them are grounded in a simplistic and rather dated perception of computers as merely tools, and partly because others are single-disciplinary efforts that sometimes fail to take into account the cross-disciplinary and cross-cultural nature of modern ICTs. New forms of expression and communication made possible by digital information and communication technologies are not just reproductions of existing forms of communication that can be studied with the same old theories and methodologies. Most of them are forms of expression in their own right with their own unique qualities and traits. For instance, an email is not the same as a traditional letter. Although we can find many similarities, they are, notwithstanding, different in many ways. "Talking" online in a MOO or a chat room may at first glance appear like a typed phone conversation, but in fact it's not. Again, we can identify certain similarities, but there are also fundamental differences both with regard to social setting, etiquette, and purpose. A web page may perhaps look like any other document, but upon closer scrutiny we'll find that it is very different from the traditional printed page in most respects. Electronic forms of text and literature, such as for instance hypertext, are expanding our notions of what text and literature are and can be. Similarly, the immensely popular genre of computer games is simply not just a modern day version of traditional folk tales and board games. Most modern computer games represent expressions and aesthetics of a different register that are quite unique in their own right. All these new digital forms of expression beg scholarly attention and intellectual analyses as to what they say about the socio-

technical processes in which they are shaped and used, and about the impact they have upon the evolution of cultural structures. We need to theorize in order to understand what they are, how they influence the way we relate to one another, and how they impact the human activity that is the focal point of the humanities. This will happen best if we go about studying these phenomena in a concerted and fully cross-disciplinary manner where theories and methodologies not just from the humanities but also from the sciences come together to infuse a common research effort.

When I first joined the Department of Humanistic Informatics in 1996 as a visiting assistant professor, I knew immediately that it was the perfect place for me to conduct precisely this kind of cross-disciplinary research and development. Before I continue with a discussion of the third and most significant challenge that underlies this project, therefore, let me briefly explain the field of Humanistic Informatics as a way of establishing the academic context in which I have conducted my work.

Humanistic Informatics

The term informatics refers to the study of Information Technology (IT). In a narrow sense it is equivalent to the use of the term computer science in North America where computer programming, systems design and analysis form the crux of the academic activities. In a broader sense the word informatics also encompasses the study of IT systems and the effect and function of said systems in society and culture. Humanistic informatics approaches this broader study of IT from a humanities perspective.

The Humanistic Informatics program at the University of Bergen was first established by professor Jan Oldervoll and professor Roald Skarsten in the mid-

1980s. It was originally called "Information processing for humanities students," and its purpose was to teach students from the humanities about how to use computers as tools in their studies and research. In 1992 Espen Aarseth joined the program as a research fellow, and he began to offer new courses that dealt with the cultural aspects of IT. His main interests at the time lay in literature and computer games, and the work he did for his dissertation, *Cybertext*, came to form much of the basis for a new direction for the program. Today, Humanistic Informatics is a program that attempts to bridge the gap between the sciences and the humanities by approaching the ICT challenge from the threshold that exists between them. The overarching aim is to make students aware of the complex relationships that exist among technology, society, and culture. It is important for students to learn about the driving forces of technological development and change and how they influence our lives and shape the conditions of our future. An equally important goal is to teach students how cultural and social structures influence and shape the technology. Thus, on the one hand, a main goal of Humanistic Informatics is to equip students with the critical and informed theoretical background they need in order to analyze and understand today's cultural and societal transformations. On the other hand, Humanistic Informatics also aspires to give students the technical knowledge and proficiency they need in order to become players on the stages where information technologies are conceived, developed, shaped, and used. This means that students must also learn computer programming in order to master the tools with which information technologies are built.

Innovate!

The third and most important challenge that, in effect, became the main thrust for this project pertains to our own role in the creation of our technological future. Although I understand that technological development can often seem inevitable and ever-progressing, I do not subscribe to the deterministic view that there are intrinsic forces at work that somehow unfold according to their own logic or at the hands of the technologists that set them in motion. To the contrary, not only do I believe that we have the power to shape our socio-technological future, but I also believe that we have an obligation to do so. Due to the massive influx of ICT in our society and the deep-seeded transformative powers that are at play, we can no longer afford the luxury of saying that technology does not concern us. Nor can we remain just reluctant users and critics of technology. As students and scholars in the humanities we must begin to actively bring our own values and experiences to bear on the technical development that is taking place, and thereby help create better technologies and a better future.

One way to assert our influence is to enter into partnerships with computer scientists and work with them to produce the technologies we want. In order to do so effectively, however, we need to possess a thorough understanding of how the technology works and how it is produced. In Humanistic Informatics, it is a requirement that students learn how to program a computer. The basic philosophy is that just as the anthropologist must master the language and customs of the culture she is studying, a student of ICT needs to understand similar traits of modern technoculture. Only by looking inside the black box of technology and actually understanding what goes on inside it can we hope to influence the technical development. Only by "speaking their language" can we hope to communicate productively with technologists.

The other way we can contribute to the technological development is to get completely involved and do the development ourselves. Although this should not necessarily be a goal for every student in the field, I felt that it was the appropriate way to approach the subject of this dissertation. Before I go on to discuss the particulars of how this doctoral project was designed, it is necessary to first set the stage by discussing one specific example in which we in the arts and humanities can contribute to new ways of thinking in design and development in order to produce better and more productive technologies for the future.

Microsoft Word is without a doubt the most commonly used word processing program in the world today. With each new version, the Word engineering team adds new features and options that they think writers want. Evidence to this effect is found in the plethora of toolbars that Word sports. Many of the features found in Word are arguably options that most writers want and from which they do indeed benefit. For example, a good system for managing footnotes and endnotes is something that every academic writer needs. Other features, such as annotation and commenting, promote collaborative writing and editing, which I believe is a good thing. Still, if we look at Word as a writer's tool, it is striking how it in fact eludes the very essence of the writing process. If we accept that writing also constitutes a form of thinking and that thinking is a complex and inherently multi-directional process, then it should follow that writing itself is not a linear activity. Thus, a system that presupposes linearity by forcing the writer to think linearly in crafting her texts does not do a good job of supporting the fundamental nature of writing. Based on this observation, one might argue that today's word processor is in fact nothing more than the latest installment in

a long evolutionary chain of linear writing tools that include such technologies as the typewriter and the printing press.

As an alternative to Word and other popular word processors, I would like to highlight the hypertext-writing environment, Storyspace. This system does not impose directionality on either the creative writing process or the textual end result. The process of creation is allowed to follow the meanderings of the mind as opposed to being dictated or constrained by the convention of linearity. Thus, I believe Storyspace is an excellent example of a technology that adapts to the nature of a particular human activity rather than trying to bend the substance to fit a certain technological or ideological dogma. An important reason why I believe Storyspace accomplishes this vital goal is that writers and humanities scholars, such as Michael Joyce, Jay David Bolter and others, were actually involved in the design and implementation of the system. Because they were writing professionals, they knew what they wanted; and by being directly involved in the technical development, they were able to translate their needs into actual software that they could put to productive use.

The Design of this Project

Starting in early 1995 I began experimenting with educational applications of an online multi-user chat and gaming technology called MOO (MUD Object-Oriented). Together with professor Cynthia Haynes of the University of Texas at Dallas (UTD), I established a MOO site named Lingua MOO for use in freshman composition, rhetoric and writing classes at UTD. When I started teaching in the Department of Humanistic Informatics at the University of Bergen, Norway the following year, I expanded the use of Lingua MOO to my own classes on object-oriented programming and Internet culture. Over the course of the first two

years of Lingua MOO we gained a lot of practical experiences with the MOO technology, and we began to identify areas in which it could be improved in order to facilitate better conditions for learning and also accommodate new technical possibilities that we wanted to offer our students.

MOO technology was not a commercial product where we could expect improvements and updates on a regular basis. In fact, the whole technology was the result of an informal and voluntary collaboration between a group of hackers and programmers associated with the online community LambdaMOO. While they were primarily interested in the social interaction and gaming aspects of MOO, we were interested in it mainly from an educational point of view. It was therefore clear to us that if we wanted to make the changes and modifications we felt were necessary, we had to do it ourselves. With any kind of proprietary technology this would have been impossible, but MOO was not proprietary, it was Free Software.

Thus began the project that is the basis for this dissertation. Right from the start it was a two-pronged effort to, on the one hand, study, analyze, and understand the history, methodology, and practices behind the development of the free software systems that we were working with, and on the other hand, to use this knowledge to implement our ideas in the form of a new and improved educational MOO system. Throughout the whole project these two activities went hand in hand. The theoretical and methodological knowledge that I derived from the study of the history of Free Software and Open Source was, in effect, the underpinning for the creation of the software program, enCore.

While Dr. Haynes continued to facilitate the daily operation of Lingua MOO, such as taking care of pedagogical matters, spatial layout design, teaching schedules, accounts, and so forth, I focused my attention on the development of

the new enCore MOO database as a general-purpose educational web application. This study is, therefore, an *enactment* of both the theory and practice of the challenges I have outlined above, as well as the implementation of the knowledge in the form of a free and robust software program. The study *enacts* what it traces historically, and it is an *enactment* of the collaboration that lies at the heart of the Open Source movement. Without the perspective of enactment, its key components would, separately, lack the force with which its mission is pursued. With such a perspective, the study is able to cohesively blend history, theory, practice, and implementation together with the diverse and numerous communities of users, students, teachers, and researchers for whom the project was designed to benefit.

How the Study Unfolds

To give the reader a better understanding of what lies ahead, let me now give an overview of the case studies that occupy the main chapters that follow.

Chapter 2, *Inventing the Future*, was designed to be a historical backdrop for the case studies of Free- and Open Source software. It is a brief historical overview of some important milestones in the history of software. Starting in the early 1800s, it traces the roots of the concept of software from the Jacquard Loom through Ada Lovelace's theoretical work on algebraic coding and control systems for Charles Babbage's mechanical calculators to Konrad Zuse's work on the Plankalkül, a sophisticated algorithmic programming system that many historians now claim was the world's first high-level programming language. I then go on to discuss the circumstances surrounding the conception of the paradigm often referred to as modern computing. The invention of the modern computer, and the so-called stored-program concept, during and immediately

following World War II created the first real possibility for a large-scale implementation of the notion of software. In the late 1940s and 50s a growing number of mathematicians became interested in the problems pertaining to programming and control of digital computers. The upshot of these efforts was the creation of automatic programming systems, programming languages, and subsequently the formation of a whole new academic field that we now know as computer science.

Research scientists and other computer professionals conducted most of the early developments in modern computer science and technology, but from the early 1960s another group of actors also began to make contributions in these areas. They were, for the most part, students and other self-taught computer enthusiasts, and they referred to themselves as the hackers. Since the 1960s, the hackers have made numerous significant contributions to computer science and to the technological reality in which we live. They played an important role in the development of the personal computer, they were instrumental in the creation of the technologies that forged the Internet, and they have been the principal architects behind some our time's most widely used network-based communication technologies.

Chapter 3, *Free as in Freedom*, is a study of their role in the Free Software movement from the 1970s up through the early 1990s. It focuses on the creation and evolution of Unix, and Unix-like operating systems, which became the focal points for some of the hacker movement's greatest achievements during this time. The analysis opens with an account of early time-sharing systems, which then leads into a discussion of Ken Thompson's creation of Unix in the early 1970s. For many years the Unix source code was freely available, and at the University of California at Berkeley a group of programmers and hackers took

this opportunity to create their own flavor of Unix named the Berkeley Software Distribution, or BSD for short. The BSD project became an important playground for experimentation with some of the collaborative development methodologies that would later become the hallmark of the Free Software movement. It is also important to note that BSD was a vital player in the creation of the computer network that we now know as the Internet.

In 1983-84, the Free Software ideology and the methodologies that had been implicit in the hacker movement more or less since its very beginning in the late 1950s (cf. Levy) received a significant boost when Richard Stallman founded the GNU project for the purpose of building a free Unix-like operating system. Stallman's great ambition was to create a new framework for software development that resonated with the old ideals of the hacker movement and its ethics—a system that was free for anyone to play with and make modifications to. To this end he instituted an organization named the Free Software Foundation (FSF) and created a whole new Free Software license dubbed the GNU General Purpose License that was specifically designed to protect the freedoms in regard to software that he believed all true hackers craved.

By 1990 the GNU system was slowly coming together. Stallman and a band of volunteer hackers loosely associated with the Free Software Foundation had assembled many of the essential components needed to build an Operating System; but there was one crucial feature missing, they did not yet have a kernel. The solution to that problem would come from an unlikely source, and its effects set the hacker movement on fire. I am referring, of course, to Linus Torvalds' operating system, Linux. The last part of chapter 3 is an investigation of the early history of Linux and the new collaborative Internet-based development model that Torvalds pioneered.

Chapter 4, *We Changed the World*, frames as its subject matter a few key historical moments of the hacker movement in the 1990s. Thanks in large part to the increasing proliferation and adoption of the Internet during this decade, Linux became the unequivocal flagship of the hacker movement. Not only did it capture the imagination of hackers everywhere, it also reeled in the popular media, and helped create a whole new Free Software economy. The Linux distribution company, Red Hat Inc., became one of the biggest players on the dotcom scene, and its business model was copied by start-ups and established companies alike. Free Software had become a hot commodity, and everyone, it seemed, wanted a piece of the action.

In 1998, at a time when the dotcom economy was still red hot, a group of hackers under the leadership of Eric S. Raymond and Bruce Perens began to voice concern that the notion of Free Software, coupled with Richard Stallman and the Free Software Foundation's ideological attitude toward software engineering, would hamper the adoption of hacker methodologies in important sectors of business and industry. Netscape, perhaps the most prominent dotcom of the decade, had just announced that they intended to release the source code to their wildly popular web browser, Netscape Communicator, and Raymond and Perens felt that this presented a unique opportunity for the hacker movement to gain even wider acceptance for its methodologies and practices. In response to Netscape's announcement, therefore, they came up with the concept of Open Source as a less dogmatic alternative to Free Software. The creation of the Open Source Initiative in 1998 seeded a deep conflict between what had in effect become two flanks of the hacker movement, and a war of words ensued between the ideologists and the pragmatists within the movement.

While the hacker movement's leaders argued over ideology, new Free- and Open Source software projects continued to emerge at an ever-increasing rate. The last part of this chapter examines two of the most seminal of these projects, the Apache web server and the Mozilla web browser, and the circumstances surrounding their development. It also examines three different online communities that became important support structures for the evolution of the hacker movement in the latter part of the 1990s.

Chapter 5, *From Code to Community*, is a study of a different type of hacker community, LambdaMOO, and the technology it produced. Most of the ideas and concepts that ultimately became LambdaMOO were derived from hacker programming efforts dating back to around 1980, where the principal aim was to build online multi-player games known as MUDs (Multi-User Dungeons). LambdaMOO represents a hacker technology that was created in true collaborative fashion. To participate, programmers actually had to be logged in to the LambdaMOO system and work on the code from within. The community was contained within the software itself, and this raises several interesting questions in regard to differences and similarities between the synchronous hacker community of LambdaMOO and the more traditional asynchronous hacker communities that developed the technologies I discuss in previous chapters. In addition to being a case study on hacker methodologies and practices in synchronous environments, chapter 5 also serves as an introduction to the MOO technology that I have used as the basis for the development of the enCore software.

Chapter 6, *Between Theory and Practice*, differs from the preceding chapters in that it reports directly on the design, development, and programming I have done in connection with the enCore Open Source Project. While the earlier

chapters were all designed to illuminate and analyze methodologies and practices fostered by the hacker movement over the past 30 to 40 years, chapter 6 tells the actual story of my own implementation of those methodologies in the enCore project. As a means of relating the project to its technical and educational context, the chapter opens with an examination of the development of the academic and educational MOO in the early 1990s and my own involvement in this process through the Lingua MOO project.

The middle part of the chapter focuses on the initial conceptualization and early implementations of the enCore software from around 1997 onward. These first development efforts were undertaken in conjunction with a book project on educational MOOs called *High Wired*, a collection of essays I co-edited with Dr. Haynes. From 1998-99 on, the enCore project expanded in scope as I set about to design a new web-based MOO client along with a new graphical user interface called Xpress. For this purpose I created an Open Source Project using the historical case studies of previous chapters as guidelines. The first version of the new enCore Xpress system was released in April of 1999, and an in-depth explanation of its many features is provided toward the end of the chapter. Some closing thoughts and remarks on the success and disappointments from the enCore Open Source project rounds out the chapter.

In addition to these main chapters, this text also includes three appendices that contain additional material relevant to the dissertation. *Appendix A* contains the enCore Manifesto. *Appendix B* is a comprehensive portfolio of the various forms of outgrowth of the enCore project, including sample lists of enCore-based MOO sites, third-party research publications and grants, as well as awards and media recognition the enCore Open Source Project has received over the years. *Appendix C* contains the text of the Open Publication License version 1.0.

Terminology

Throughout this text I am using several key terms and expressions whose meanings may or may not be immediately clear to the reader. In the interest of clarity, and to avoid possible confusion, let me therefore take a moment to explain the most important of these terms.

Hacker. According to Eric Raymond's New Hacker's Dictionary, the term hacker was "first adopted as a badge in the 1960s by the hacker culture surrounding TMRC [Tech Model Railroad Club] and the MIT AI Lab." ("New Hacker's) The term has acquired many meanings over the years. Here are some of the most common definitions.

> 1. A person who enjoys exploring the details of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn only the minimum necessary. 2. One who programs enthusiastically (even obsessively) or who enjoys programming rather than just theorizing about programming. 3. A person capable of appreciating hack value. 4. A person who is good at programming quickly. 5. An expert at a particular program, or one who frequently does work using it or on it; as in `a Unix hacker '. (Raymond "New Hacker's")

In the 1980s the word hacker also came to be synonymous with someone who used their extensive programming skills and knowledge of computer systems to gain illegal access to computer networks, databases and all sorts of sensitive information. According to Richard Stallman, however, this is a misappropriation of the word. He explains: "The use of 'hacker' to mean 'security breaker' is a

confusion on the part of the mass media. We hackers refuse to recognize that meaning, and continue using the word to mean, 'Someone who loves to program and enjoys being clever about it'" (Stallman, "The GNU Project"). Throughout this study I have used the word hacker consistent with its original meaning.

Am I a hacker? I don't know. I certainly share their fascination for computers and computer programming. Through this project I have also practiced many of the software development methodologies that they have fostered, but all this is inconsequential. Like a doctorate, hacker is an honorary title that must be bestowed upon you by your peers. No true hacker would ever claim it for himself.

Hacker movement and hacker community. These terms are used somewhat interchangeably throughout the text. Eric Raymond uses the phrase 'hacker community' to describe the broader fellowship of hackers everywhere (Raymond, "A Brief History"); but from a historical perspective, talking about one overarching hacker community can be somewhat misleading. During the early days of hackerdom, hackers usually congregated in smaller, individual communities connected to local research institutions and universities. While they may have shared common interests and traits with hackers in general, their ties and loyalties were first and foremost directed at the local level. As the computer networks started to spread within academia in the 1970s and 80s, we start to see a growing globalization of the hacker communities and the formation of a hacker movement. Electronic bulletin boards, email, and news groups created new meeting places where hackers could gather and form communities across many different boundaries. The creation of the GNU project and the Free Software Foundation in the mid-1980s represents an important step toward the consolidation of this movement. Linux and other seminal hacker projects, as well

as the establishment of the Open Source Initiative in the 1990s, are further evidence that the hacker communities during this time had joined forces in a movement to spread hacker methodologies, practices and ethics to the world at large.

Free Software and Open Source. Both terms refer to a set of principles for software development as fostered and practiced by the hacker movement. These principles can be summed up as follows:

- Users should have the freedom to utilize a computer program for whatever purpose they wish.
- They should always have access to the complete program source code.
- They should be able to make modifications and/or additions to the source code if they so desire.
- They must be allowed to redistribute their modifications as derivative works either *gratis* (as differentiated from free, as in cost) or for a fee.

Although Free Software and Open Source may be concerned with the same practical goals, they do represent two rather different philosophies on software engineering within the hacker movement. Richard Stallman, founder of the Free Software movement, puts it this way.

> The fundamental difference between the two movements is in their values, their ways of looking at the world. For the Open Source movement, the issue of whether software should be open source is a practical question, not an ethical one. As one person put it, "Open source is a development methodology; free software is a social movement." For the Open Source

> movement, non-free software is a suboptimal solution. For the Free Software movement, non-free software is a social problem and free software is the solution. (Stallman, "Why Free Software")

In this project I have chosen to use the phrase Open Source both in regard to the enCore software project itself and as an overarching label for the hacker methodologies that I have studied. The concept of Open Source is practical and non-committal, and one can use it without subscribing to the more abstract moral arguments that are embedded in Free Software.

MUD and MOO. These are acronyms that stand for Multi-User Dungeon and Multi-User Dungeon Object-Oriented, respectively. MUD was initially the name of the original text-based multi-user adventure game developed by Richard Bartle and Rob Trubshaw in the late 1970s. A more detailed history of MUD is found in chapter 5. Since then, the MUD system has been reverse-engineered and expanded numerous times so that it has in effect become a generic label for many types of text-based multi-user adventure games. Thus, a MOO is a type of MUD that has been expanded and modified from its original form. Some of the most prominent features that distinguish MOO from other MUDs include a user-extensible, object-oriented database, a built-in programming language, and a general lack of traditional adventure game and role-playing features.

Research Methods

As mentioned earlier, I have situated this research on the threshold that both separates and binds together the humanities and sciences, and that bridges the gap specifically between the various fields encompassed by the humanities and informatics. Thus, the cross-disciplinary nature of this dissertation made it not

only necessary for me to draw upon research methods from both history and Open Source; it called for inventing methods and forms that heretofore could not be categorized easily nor definitively identified as such. Innovation is derived from the impetus to invent something 'novel,' or new. It also frequently necessitates new 'forms' that in their infancy can only be described as 'hybrid' until such time as future research advances more specific terms with better-articulated meanings. Since the software development methodologies that I have used are comprehensively covered in the following chapters, my final introductory remarks concern the historical research methods that inform this study and the hybrid nature of the method and 'form' constituted by this manuscript and the accompanying enCore software distribution.

In order to lend force to the leading edge I sought both to trace and *enact*, it was important that I conduct my research via the media I studied and the communities they represented and the networks in which they developed. Thus, the research for the historical case studies in chapters two through five was conducted almost entirely over the Internet. The bulk of the archival material, including both primary and secondary sources, was found using Google, the most widely used search engine in the world today. An important reason why Google is also an excellent research tool is its unique web page ranking system, PageRank, developed by Stanford University graduate students Sergey Brin and Larry Page. In a paper entitled "The Anatomy of a Large-Scale Hypertextual Web Search Engine," presented at the Seventh International World Wide Web Conference in Brisbane, Australia in 1998, Brin and Page explains:

> Academic citation literature has been applied to the web, largely by counting citations or backlinks to a given page. This gives some

> approximation of a page's importance or quality. PageRank extends this idea by not counting links from all pages equally, and by normalizing by the number of links on a page. (Brin, "The Anatomy")

In this manner Google assigns a "page rank" to each page that it encounters on the web. If a certain page is "cited" by many other pages, its page rank will be higher than if it is "cited" by only a handful of pages. In addition, when assigning page ranks, Google also takes into account the fact that some web sites are of higher quality and/or importance than others. Pages residing on "important" sites will therefore gain an additional ranking bonus. These are the main features that have made Google such a popular and efficient tool for information retrieval; but, incidentally, they are also the reason why some people are voicing concern about it (Brandt). Much of the worry centers on what people feel are the "un-objective" and "un-democratic" features of PageRank. In an ideal world one link should in principle count as much as the next, but in PageRank, as I have just explained, pages are not treated as equal. The ranking system might give an unfair advantage to popular pages residing on popular sites, and it may well give Google as a company too much power and control over information on the web. On the other hand, Google and its unique PageRank system do offer the best and most reliable in web search technology available today, and in my work here it has also proved highly useful as a tool for quality assurance of sources, which is something with which every historian must always be concerned. Since most of the software technologies I have studied have been related to the Internet in one way or another, I did not have any problems identifying authentic primary source material on the web. In fact, over the period of time that the historical research was conducted, I noticed a significant increase in the number

of older primary source materials being put online. In addition to the written source material, I also conducted four interviews with a few of the principal figures that appear in this study. Because I have been personally involved in some of the events covered in this study (see chapter 6), I have also been able to draw on my own research archive, which, among other things, includes the complete enCore mailing list dating back to 1997.

In addition, it is useful to point out that the inspiration for the *High Wired* volume was grounded in the necessity of collaboration *as* method, something that is more frequently condoned in the sciences than in the humanities, but is becoming more legitimate in the humanities as the Internet both affects and shapes the production of knowledge in its diverse fields. As an example of this phenomenon, it was with great difficulty that we conceived a subtitle for the book. Our editors wanted to be able to market it effectively, and to do so they had to understand the hybrid nature of its content and mission so that our readers would also. After much discussion, we agreed to subtitle it, *On the Design, Use, and Theory of Educational MOOs*. In retrospect, the subtitle itself may not have been an important factor in the book's success, but the fact that it is now in its second edition suggests that their gamble on this 'book' that contained poetry, essays, personal narratives, dense theory, and practical applications, as well as a fully developed and free educational software program, paid off. To further capture the 'novel' nature of the enCore project, which even more directly speaks to the reason for the form that this dissertation takes, it became apparent that as new forms of research methodologies were required to develop and disseminate the results of the work, it was increasingly apparent that conventional *methods* of writing were transforming traditional *forms* of writing as well. The writing I was doing as a scholar in 'humanistic informatics' needed to

be defined in such a way that it included not just theory and practice, interpretation and analysis, it also needed to include programming. Thus, in the textbook, *MOOniversit*y, it was my aim to broaden the notion of writing to contain more than the forms we know as novels, essays, poetry, and other genres. Writing is thinking, and if programming is also writing, then it is thinking put to its most ardent test—application. For the first time, a writing textbook, adopted primarily by college-level writing programs, included a chapter on *programming* as writing.

In the wake of all this, it seems not only natural to include the programming with the manuscript of this dissertation; it is fundamental to the rendering of its hybrid nature, and crucial to its having been situated within something called, for now, *humanistic informatics*.

The enCore Software Distribution

As I have argued, the enCore Open Source distribution is in itself a key element of the whole project. It includes the latest enCore system along with all the updates that have been released since 1997. In addition, it contains binaries and source code for Pavel Curtis' LambdaMOO server, plus the complete source code for the Xpress client and GUI layer that I wrote in connection with this project. The MOO code for the remaining enCore base system can be viewed from inside any enCore MOO.

You may use the enCore software to set up a complete enCore MOO. Setup instructions for Unix-based systems are included in the distribution. Alternatively you may explore the enCore Xpress system from a user perspective by connecting to Lingua MOO at http://lingua.utdallas.edu:7000/. You are

Directory	Sub directory	Contents
encore		Contains version 3.3.3 of the enCore distribution.
	images	Contains all the icons and artwork that were developed for enCore version 3.0.
	texts	Contains Pavel Curtis' MOO Programming manual.
	sounds	Contains the sound files used by the Xpress system.
	vase	Contains files specific to the Virtual Assignment Server Environment (VASE).
	stylesheets	Contains style sheets used by Xpress if so enabled.
	mootcan	Contains binaries, source and documentation for Sindre Sørensen and Lingo.uib's MOOtcan telnet applet.
updates		Contains 30 updates to the enCore system released between 1997 and 2003. Source code.
bin		Contains binary versions of the LambdaMOO server version 1.8.1 for the following operating systems: Mac OS X, Linux, Solaris, Solaris X86, and DEC 4.0.
scr		Contains the MOO source code for the enCore Xpress client and GUI layer.
MOO-1.8.1		Contains the source code for the LambdaMOO server version 1.8.1.

welcome to request a user account in LinguaMOO, or connect as a temporary guest.

Perhaps viewing the contents of the enCore software distribution in advance of reading the text that follows will help readers understand the scope of the

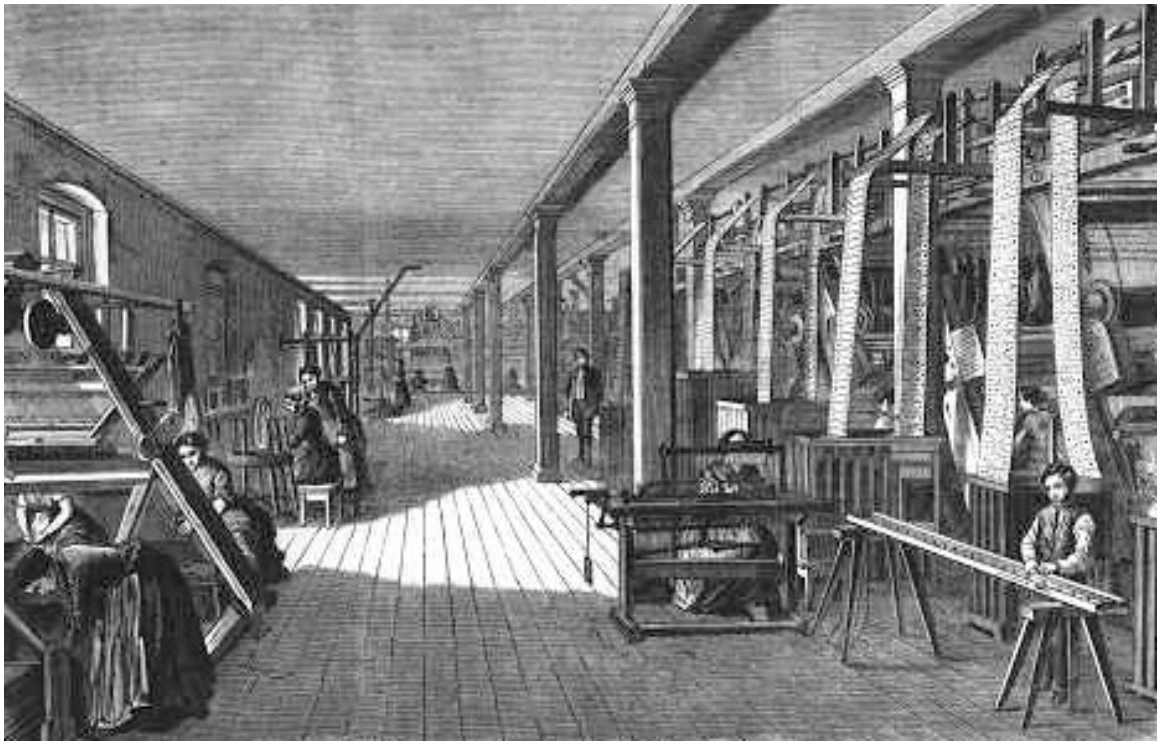
dissertation. But reading it last could also serve to cap off the historical and investigative, personal and practical, aspects of reading the text contained in the chapters. Either way, this story can begin as easily in 1994 as it stands now in 2003—an enCore of an altogether different performance in writing, an enCore that is a hybrid of both preface and epilogue.

2

To Invent the Future
A History of Software

The best way to predict the future is to invent it. Really smart people with reasonable funding can do just about anything that doesn't violate too many of Newton's Laws! –Alan Kay, 1971

Early attempts to "program" machines to perform automated tasks can be traced back at least to the industrial revolution of the late seventeenth and early eighteenth century. The perhaps best-known example of such an early programmable machine is the *Jacquard Loom* invented by the French silk weaver Joseph-Marie Jacquard (1752-1834) in 1801. The Jacquard Loom was an automatic weaving machine that could be programmed to automatically weave any pattern in fabric. Data about various patterns was encoded as holes or the absence of holes on pasteboard cards. When a series of these cards was fed into the machine, it would mechanically read the punched cards and automatically reproduce the patterns that had been encoded onto them. Some of the most intricate and complex patterns woven by Jacquard Looms could take tens of thousands of cards to produce.



The Jacquard Loom. Wood engraving from 1858. Courtesy of the Deutsches Museum.

While the Jacquard Loom was a programmable machine that bore no resemblance to calculating machinery whatsoever, another early example of a programmable machine became, in many historians' opinion, the very precursor to the modern computer itself.

In 1833 the British mathematician Charles Babbage (1791-1871) devised a sophisticated mechanical calculator that could be programmed to automatically perform scientific computations. Babbage called his machine the *Analytical Engine*. The ideas for the machine grew out of another project, the *Difference Engine*, which he had worked on since 1823. In his autobiography, Charles Babbage describes how his life-long pursuit of automatic calculating machinery began.

> One evening I was sitting in the rooms of the Analytical society at Cambridge, my head leaning forward on the table in a kind of dreamy mood, with a Table of logarithms lying open before me. Another member, coming into the room, and seeing me half asleep, called out, "Well, Babbage, what are you dreaming about?" To which I replied, "I am thinking that all these Tables might be calculated by machinery." (Williams 163)

Unfortunately, due to the limitations imposed by the tool making art of the early nineteenth century, and problems of funding caused in part by his somewhat eccentric personality, Babbage was never able to successfully complete the Analytical Engine. Thanks in large part to the writings of Lady Ada Lovelace, daughter of the illustrious British poet Lord Byron, however, the ideas and concepts of the machine were passed down through history.

Ada Lovelace had become interested in mathematics at an early age, and after she met Babbage in 1834 and learned of his work she too became fascinated with his research into automatic calculating machinery. She was one of the few people at the time who were able to understand and explain Babbage's work. In one of her many notes she eloquently sums up the concept of the machine with the following words: "We may say most aptly that the Analytical Engine weaves algebraic patterns just as the Jacquard-loom weaves flowers and leaves" (Toole 696). Ada Lovelace is also credited with being the one who came up with the idea for an algebraic coding system with which to "program" the still-theoretical machine. Her theoretical thinking went far beyond just mathematical calculations, however, something that is clearly evidenced in another of her notes.

Ada Byron, Lady Lovelace (1815-1852)

Again, it [the Analytical Engine] might act upon other things besides numbers, were objects found whose mutual fundamental relations could be expressed by those of the abstract science of operations, and which should be also susceptible of adaptations to the action of the operating notation and mechanism of the engine… Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent. (Toole 694)

Ada Lovelace's significant theoretical achievements in the area of machine programming has led some to erroneously dub her the world's first computer programmer (see for instance Rheingold *Tools*). Although there are conceptual similarities between Lovelace's work and the computer programming systems devised in the twentieth century, there is no historical evidence to suggest any direct linkages between the two. Nevertheless, in the history of computing machinery, especially as it relates to the art of programming, Lady Ada Lovelace remains one the most prominent figures of all time. In 1979, the U.S. Department of Defense named a new programming language Ada in her honor.

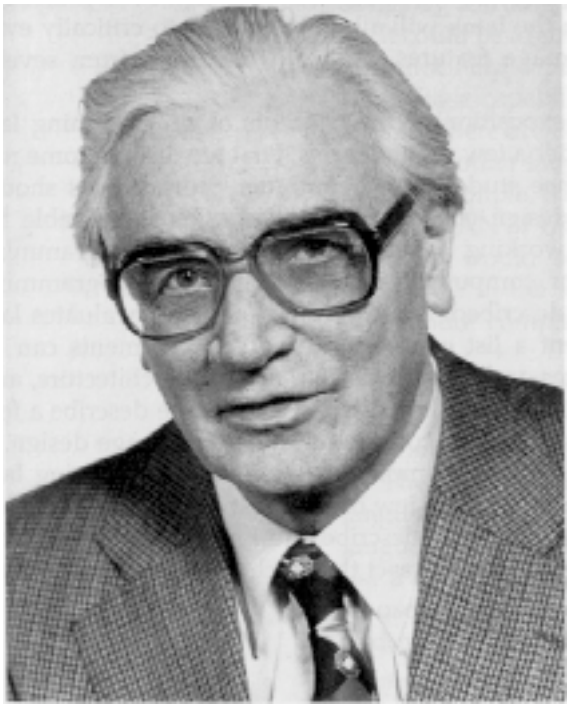
A Universal Machine

The modern computer, unlike computing and data processing machinery of previous eras, was a universal machine. It was essentially capable of solving *any* problem that could be adequately conceived of and described in an algorithmic manner. In 1937, the British mathematician Alan M. Turing published a now famous paper entitled *On Computable Numbers* in which he formulated many of the modern computer's theoretical underpinnings. In this paper Turing addressed the question of unsolvable mathematical problems known as Hilbert's Entscheidungsproblem. He was able to prove the existence of such problems by devising a theoretical machine (Turing machine) that was, in effect, capable of solving any computable, that is, solvable mathematical problem. While the Turing machine and the theory behind it was an important factor in the conceptualization of the universal machine, it was not until World War II that the first prototypes of what we today consider the modern computer started to emerge.

In wartime Germany, under the cloak of Hitler's Nazi regime, a young engineer named Konrad Zuse invented what many historians now believe to be the world's first fully functional digital programmable computer (cf. Ceruzzi). The machine, known as the Z3, was based on Zuse's earlier work with electromechanical calculators dating back as early as 1934. With the outbreak of the war in 1939, young Zuse was drafted into military service, but he managed to work on the Z3 off and on until the machine became operational in 1941. As Zuse explains, German aviation authorities took an immediate interest in the Z3 and realized that it could be a valuable tool in aircraft design.

> Unlike aircraft stress, wing flutter results in critical instability due to vibration of the wings, sometimes in conjunction with the tail unit. Complex calculations were needed in order to overcome this design problem. […] I achieved a breakthrough using my equipment for this calculation. Unfortunately the Aircraft Research Institute had not been given a high enough priority for me to be released from military service. (Lee "Konrad Zuse")

Regrettably, the Z3 was lost during an allied bombing raid on Berlin in 1944, but by this time Zuse was already hard at work on his next machine, the more powerful and improved Z4. As allied forces closed in on Berlin in the late winter and spring of 1945, Zuse was forced to suspend his work in the city and move the Z4 around what was left of Nazi Germany in an attempt to avoid capture by the allies. He was successful in his escape and the machine eventually ended up in



Konrad Zuse (1910-1995). Inventor of the world's first programmable digital computer.

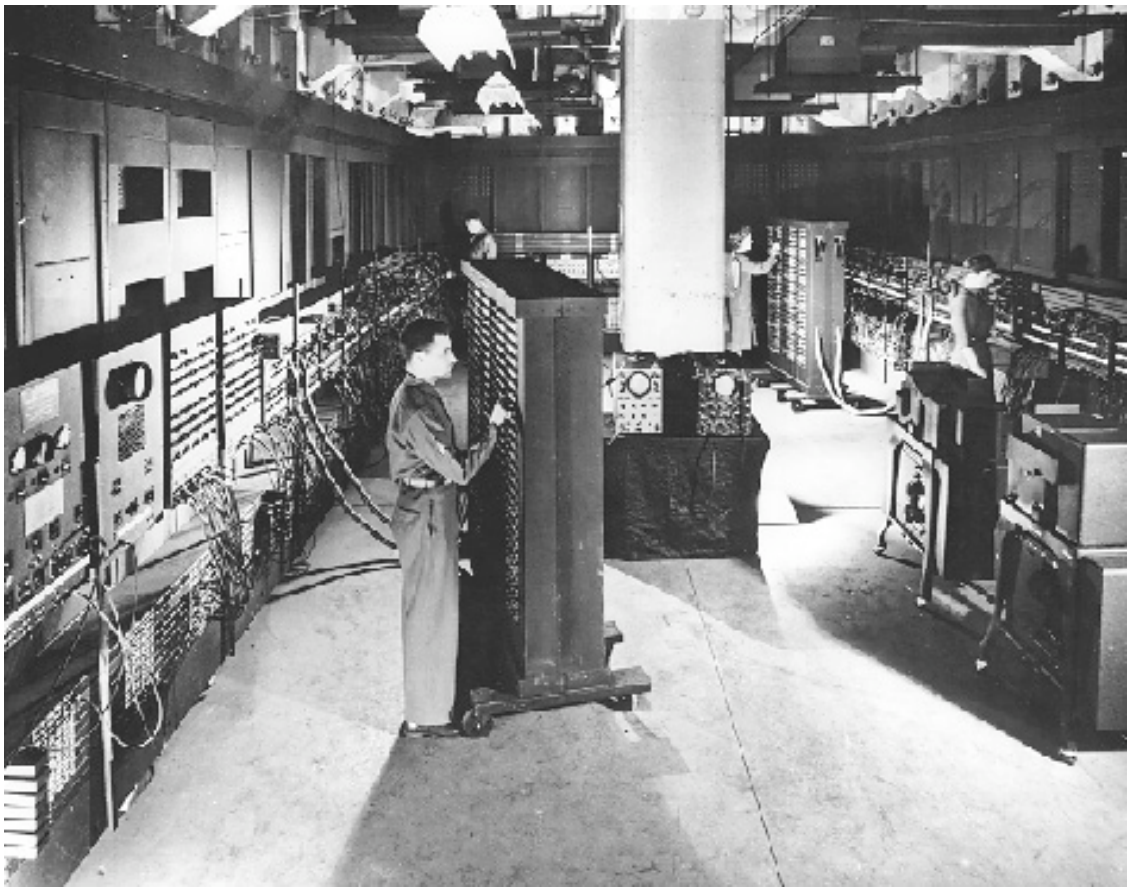
the small alpine village of Hinterstein where the next chapter in the Z4 saga unfolded. During the war years Konrad Zuse had, as we shall soon see, also become interested in problems related to the programming of his machines. He says:

> One aspect became clear to me in view of all this research between 1936 and 1946. Some means was necessary by which the relationships involved in calculation operations could be precisely formulated. My answer was "Plankalkül" - today it would be termed an "algorithmic" language. (Lee "Konrad Zuse")

Far from the inferno and the war in Berlin, in what once had been a stable in the peaceful Bavarian countryside, Zuse and his team managed to restore the Z4

to a working condition and work on the Plankalkül began in earnest. While some historians hold that Plankalkül was the world's first high-level programming language (Giloi), it was not, according to Zuse, "conceived as a means of programming the Z4 or other computers available at the time." He says; "Its true purpose was to assist in establishing consistent laws of circuitry, e.g. for floating point arithmetic, as well as in planning the sequence of instructions a computer would follow" (Lee "Konrad Zuse"). Nevertheless, Zuse did use Plankalkül for specific programming purposes. One of the first applications that he wrote using Plankalkül was in fact a chess playing program. "I remember mentioning to friends back in 1938 that the world chess champion would be beaten by a computer in 50 years time" (Lee "Konrad Zuse"). We know today that he was not much off in his prediction and that he himself had a lot to do with its realization. Although Konrad Zuse was clearly ahead of his time, his remarkable achievements were not generally known or acknowledged until many years later. Germany had lost the war, and as we know, victors write the history.

The most influential developments leading to the conception of the modern computer took place on the other side of the Atlantic, at the University of Pennsylvania's Moore School of Electrical Engineering. In the summer of 1943 the U.S. Army Ordnance commissioned a team of Moore School engineers headed by J. Presper Eckert and John W. Mauchly to build a high-speed computer for the production of ballistic firing tables. Two years later, in the fall of 1945, the team presented ENIAC (Electronic Numerical Integrator and Computer), the world's first electronic digital computer. Unlike Zuse's electromechanical machines, the ENIAC was a fully electronic computer that could crunch out numbers with far greater speed than any other calculating machine at the time. Yet, for all its speed, ENIAC had to be manually reconfigured in order to solve different



ENIAC 1945. Programming the world's first electronic digital computer.

computational problems. The machine had several great plug boards on which the "programmers" created physical binary representations of the algorithms they wanted to run. For each new program they had to go through a tedious and error prone process of physically rewiring these plug boards.

The process of building the ENIAC taught the engineers a lot about the potential of high-speed digital computers. In 1945 Eckert and Mauchly had a series of discussions with renowned mathematician John Von Neumann, of Princeton University's Institute for Advanced Study, and came up with the idea of using a "memory" to store data and computational instructions inside the machine itself (Von Neumann). This later became known as the *Stored-Program*

Concept and was the foundation for the development of all the software technologies that followed.

The Origins of Computer Programming

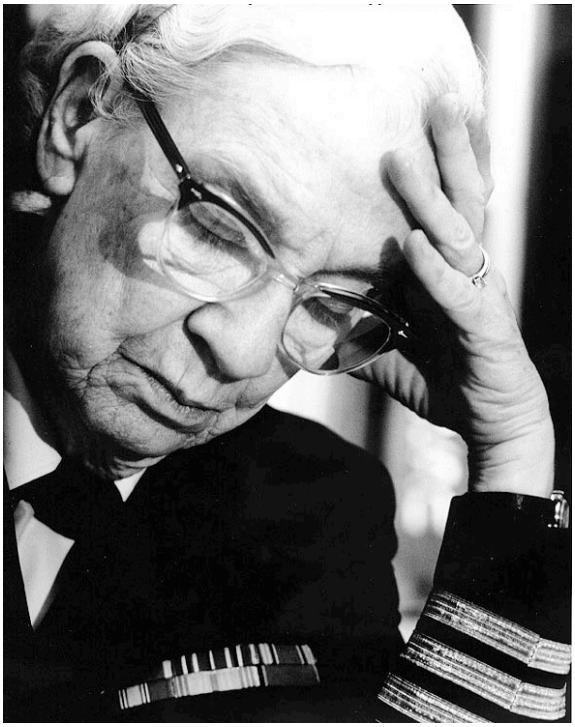
In the early days of computer programming, algorithms were typically coded directly in binary. This was a tedious and error prone process, so the first larger computer programs were instead written in octal, a numbering system of base eight which made the transition between our decimal system and the computer's binary system somewhat more manageable (cf. Hopper). As the modern computer became more widespread, a number of other programming systems were also adopted. One of the most popular of these was the assembly system that implemented mnemonic codes like ADD and MUL to substitute certain machine instructions and thereby make the code easier for humans to read and write. Another problem that frequently occurred when manually translating programs into machine-readable form was the unintentional introduction of errors. Even the slightest mistake could corrupt the entire program, and as a result, the programmer often had to spend much time finding and correcting the problem. As long as the computer was primarily being used for scientific computations, and furthermore, carefully tended to by a staff of highly skilled mathematicians doing the programming work, coding in octal or assembly, for example, was a feasible way of utilizing the sparse machine resources because of its isomorphic relationship to machine code. However, during the 1950s, as the computer was increasingly being applied to other areas such as business data processing, for instance, this scheme became more problematic.

When the Eckert-Mauchly Computer Corporation around 1950 started selling their Universal Automatic Computer, better known as UNIVAC, one of the

things they did to remedy this problem was to provide a system for easier programming. The system called Short Code was written by John Mauchly in 1949, and it was the first serious attempt to give the "programmer an actual power to write a program in a language which bore no resemblance whatsoever to the original machine code" (Hopper 9).

In 1951, the mathematician Grace M. Hopper also started to look into this problem. Hopper was already among the veterans in the emerging field of modern computing and had considerable experience in computer programming dating back to projects she had worked on during the war. Her point of view was that "you could make a computer do anything which you could properly define" (Hopper 13). So, why not let the computer automate the entire code translation process? The solution that she came up with in May of 1952 was the *compiler*, a program that took as its input another program and gave as an output a machine-readable, or binary version, of that program.

The advent of the compiler revolutionized the art of computer programming. It was able to translate code into binary both fast and accurately, and, consequently, programmers no longer had to think like a machine when formulating solutions to the problems they wanted to solve. Instead they were able to focus on the more abstract principles and mechanisms that constituted the algorithmic solution to their problems. The new "automatic" programming scheme pioneered by Hopper, combined with efforts to create pseudo-coding systems such as Short Code, eventually led to what we know as high-level programming. Another equally important "side-effect" of the compiler was that it opened the potential for commercial software. People cannot easily read programs distributed in binary form so a compiled program is therefore an excellent safeguard for intellectual property such as algorithms. Although the



Rear Admiral Grace Murray Hopper (1906-1992). Photo courtesy of The Naval Historical Center.

compiler made all this possible very early on, software development for commercial gain did not take off until at least a decade later. During these early years the money was in hardware—software was mostly an incentive to drive hardware sales.

Programming Languages

From the late 1950s onward an increasing amount of research went into the development of high-level programming languages. It created a whole new understanding of what modern computers could do, and by extension, it also spawned a whole new field of scientific research. What had previously been considered to be "applied mathematics" would henceforth be known as Computer Science.

A programming language is, in essence, a set of natural language-like statements coupled with mathematical and operational rules for how to formulate problems that a computer can solve. With the aid of such languages, the once arcane art of coding a computer became radically simplified. Problems could now be formulated using an English-like syntax where the programmer focused on the logical solution to a problem rather than the technical oddities of binary coding and translation.

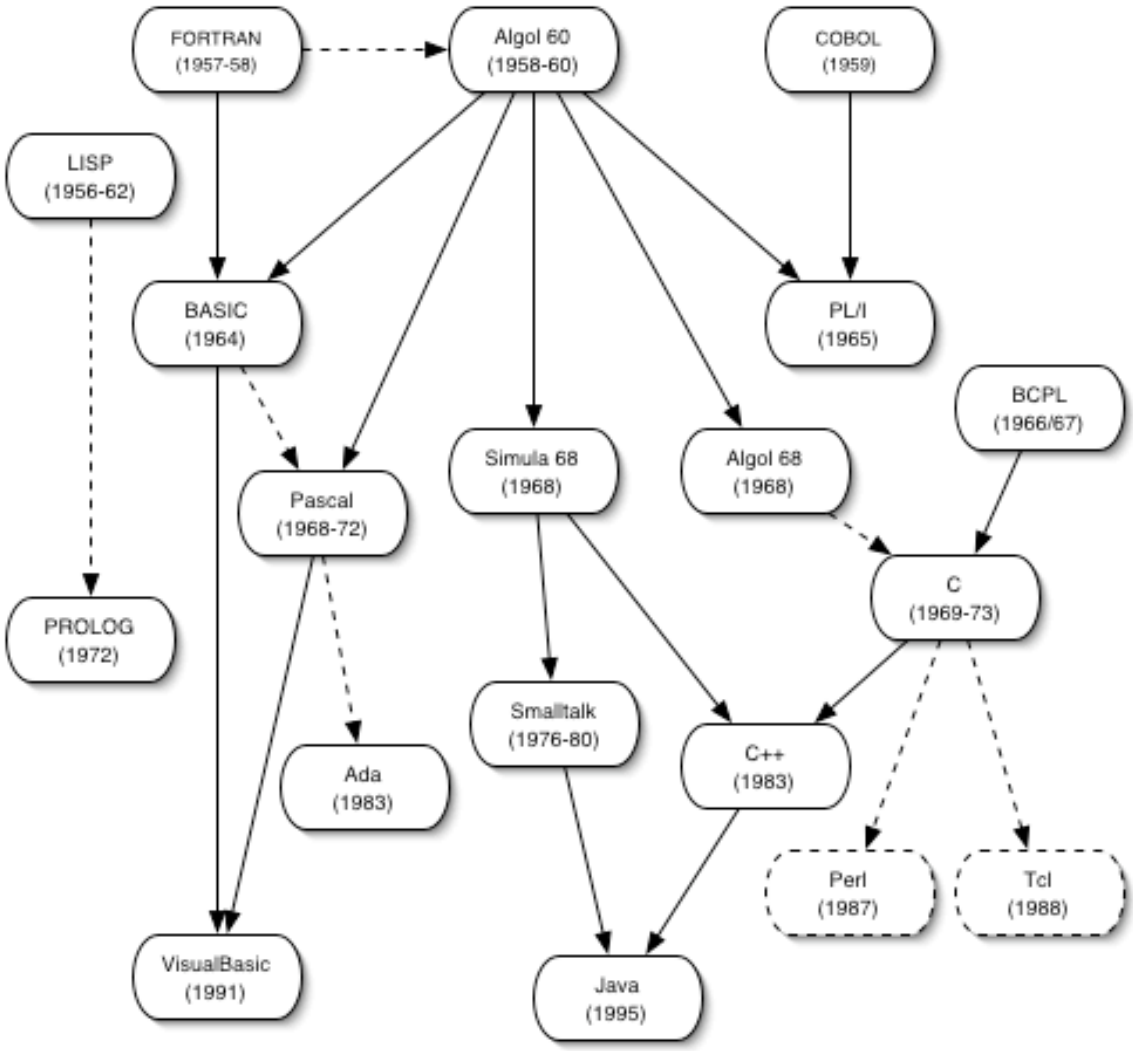
Two of the most widespread programming languages ever, FORTRAN (FORmula TRANslation, 1957) and COBOL (Common Business Oriented Language, 1959) first came into existence during the late 1950s. Another less widespread, yet highly influential, programming language named Algol (ALGOrithmic Language, 1960) was also born during these early days of high level programming language development. Together, these three languages came to form the nucleus that shaped the direction in which subsequent efforts in programming language development and software design would follow.

```
BEGIN
FILE F (KIND=REMOTE);
EBCDIC ARRAY E [0:11];
REPLACE E BY "HELLO WORLD!";
WHILE TRUE DO
  BEGIN
  WRITE (F, *, E);
  END;
END.
```

The classic “Hello World” program written in Algol.

Within a few short years there were programming languages for scientific computation, data processing, simulation, artificial intelligence and many other areas of application. While the trend early on leaned heavily toward the proliferation of special purpose languages, the search for the general purpose programming language became the primary focus of the science and engineering efforts toward the end of the decade. The desire to conceive unifying and general purpose programming concepts arose partly out of a more mature understanding of the modern computer as truly a universal machine. In part it came as a response to a notion that software development in the late 1960s had reached a state of crisis (Naur and Randell). The outcome of the debates surrounding these issues was, on the one hand, the new field of Software engineering, and on the other, a general consensus about the importance of general purpose programming tools.

The first steps that led to the idea of the general purpose programming language took place ten years earlier. Early programming language development was predominantly platform specific, i.e., IBM was making FORTRAN for their hardware, and Univac was making FLOW-MATIC (and later COBOL) for their systems. In 1958, therefore, an international committee of research scientists met in Zurich, Switzerland, with the goal of creating a universal, platform independent language that could serve as a platform for publication of algorithms. While the original Algol 58 specification was important in its own right, it was the refined Algol 60 two years later that would become one of the most important milestones in the history of programming languages. Even though Algol was for all intents and purposes a language for scientific



Milestones in the History of Programming Languages, 1957-1997.

computations, it clearly demonstrated the benefits of code portability and unifying language constructs.

When Algol became such an important milestone in the history of programming languages it had much to do with the fact that is was an international research effort. Many of the luminaries of Computer Science started their careers in the Algol research community, and several of the language projects that continued to lead the field into the 1970s and 80s had their origins in Algol. One such language was Simula developed by Kristen Nygaard and Ole-

Johan Dahl at the Norwegian Computing Center in Oslo, Norway, between 1964 and 1968. Simula was originally an Algol-based language for discrete-event simulations, but with the increasing emphasis on generality, it was later revised into a full general-purpose programming language. Simula is notable because it was the first language to introduce the concepts of Object-Oriented Programming, which became the dominant paradigm in computer programming in the 1990s. Another important language that grew out of the Algol research community was Pascal, written by Niklaus Wirth of the Swiss Federal Institute of Technology in Zurich between 1968 and 1972. Pascal became perhaps the most important educational programming language of the 1970s and 80s.

By the time the 1970s rolled around, general purpose programming languages had become the mainstay of computer science. During the next ten years, most of the development efforts in programming language design would be directed not so much toward the creation of new concepts as toward the refinement of the principles of generality. The one language that perhaps more than any other managed to capture the spirit of universal applicability was C, a language designed by Dennis M. Ritchie of Bell Labs in the years between 1969 and 1973. C was developed in close relation to the Unix operating system, and for this reason it quickly became a favorite development tool among programmers in many fields.

In the 1980s the focus shifted once again toward new theories. This time concepts such as Object-Oriented programming, modularity, and the re-use of code became the focal points for the research and development efforts. While Object-Oriented Programming had first been introduced in Simula back in the 1960s, it was languages such as Smalltalk, C++, and Java that would bring them

out into the world and establish the paradigm that dominated the field around the turn of the century.

The Hackers

By the 1960s, computers were becoming commercially available in greater numbers. Companies such as Sperry Rand Univac, IBM, DEC, and others, offered systems that were within the financial reach of universities and research institutions. For the first time students on a broader scale would get a chance to work with the new and exciting computer technology. In his book, *Hackers: Heroes of the Computer Revolution*, Stephen Levy examines how one such group of students at the Massachusetts Institute of Technology (MIT) became so fascinated with the computer that it completely consumed their lives. These were the original hackers. Contrary to popular belief, a hacker is not a computer criminal, but rather, in the words of the world's perhaps most famous hacker, Richard Stallman, "someone who loves to program and enjoys being clever about it" (Stallman "The GNU").

The first MIT hackers were young male students who shared a deep fascination for computers and the things they could do. To these young men, the computer represented something far more than a mere tool to achieve other goals. To them, the computer itself was the goal. In his study of the early MIT hackers, Levy reveals that they not only shared a common interest in computers, but that they also shared a community of values and cultural traits. Over the years the hackers have played the part of both villains and heroes. Their contributions to the history of software were not like the academic and business contributions that I have discussed above. Their approach was an idealistic one. They did not code in order to produce academic papers, programming standards,

or to make money. They coded simply because they loved doing it and because they thought it was fun. According to the *New Hacker's Dictionary*, hacking as it relates to computers might be understood as "an appropriate application of ingenuity. Whether the result is a quick-and-dirty patchwork job or a carefully crafted work of art, you have to admire the cleverness that went into it" (Raymond, "New Hacker's").

For the hackers of the early 1960s, computers were not easily accessible. The machines of the day were typically large mainframe systems that did not allow for exclusive individual use. Because of the heavy investments made in these machines, it was paramount that they be used to the maximum capacity at all times. For this reason, batch processing was the order of the day. In a batch processing system, computer programs and data were prepared off-line and then fed into a queue of tasks that the computer would then carry out. The MIT hackers viewed batch processing as an oppressive system that kept them out and limited their opportunity to use the machines they knew were available. Many of them still managed to gain access to the computers late at night, though, when no one else was around.

> During the daytime if you came in, you could expect to find professors and students who didn't really love the machine, whereas if during the night you came in you would find hackers. Therefore hackers came in at night to be with their culture. And they developed other traditions such as getting Chinese food at three in the morning. And I remember many sunrises seen from a car coming back from Chinatown. It was actually a very beautiful thing to see a sunrise, cause' that's such a calm time of day. It's a wonderful time of day to get ready to go to bed. It's so nice to walk

> home with the light just brightening and the birds starting to chirp, you can get a real feeling of gentle satisfaction, of tranquility about the work that you have done that night. (Stallman "Lecture")

The concept of time-sharing was invented in order to make more efficient use of a computer's resources, and in universities it significantly improved the students' access to computers. Time-sharing was built around the idea that users share the CPU's resources. This is done by way of an operating system that divides the CPU's total power among all users so that each user gets the feeling of having the entire machine to themself. With the advent of time-sharing, a truly interactive approach to computer programming could be adopted, and this was something that suited the hackers perfectly.

In the 1950s and 60s computer software was not generally something that was commercially available. Typically, the computer makers would commission or write software for their own systems. This software was then provided as part of the computer system that was purchased. Software was, in other words, something that was not sold separately, and as such, didn't have much of a market value of its own. The value of the software laid in the total solution that computer vendors could offer to augment their hardware systems. In many cases this meant programming languages and compiler systems that users could employ to develop their own software solutions.

Programming was a novel activity with which only a few people had much experience. In this embryonic environment, the hackers coded for fun, to see how far they could go, and how they could impress their friends and peers with ever more ingenious and elegant programs. The code they wrote was freely circulated among the members of small hacker groups, and anyone could make changes to

someone else's code. Few hackers thought of actually making a living from programming computers, or pursuing an academic career in computer science. For some observers, the hackers were wiz kids who could do things with computers few others could match; for the majority, however, they were viewed as socially inept outcasts. To the hackers none of this mattered much. In their own eyes they belonged to an elite group of people, code warriors who had conquered the mighty computer and bent it to their will.

From the 1960s onwards the hackers developed a collaborative model of software development where anything from code fragments to entire computer programs were considered community property and freely shared among its members. At the core of this gift-economy was the source code. A computer program in readable form is called source code. Anyone who knows the syntax of the particular programming language in which the program is written can usually read and understand what the program does. With the right tools and knowledge, they can change and adapt the program to their own particular purposes.

Since the computer has no concept of what the source code means, it must be translated into machine-readable binary form before it can be used. The compiler is a computer program that was developed for the express purpose of rapid automatic translation of source code into binary code. As I have mentioned, by doing this, the compiler opened up the concept of proprietary software. When the source code is sent through a compiler and translated into computer-readable binary form, it is no longer possible to read it. Nor is it possible to change it unless the source code is available. This means that the programmer can keep the source code to themselves and only give others access to the finished binary versions of programs. In the limited market for commercial software that existed

in the 1960s, proprietary software was still largely an unfulfilled promise, but things were about to change.

The Personal Computer

In 1971 Intel introduced the microprocessor—also called the "computer on a chip." With the advent of the microprocessor, the hackers' dream of having personal computers at their disposal suddenly became a possibility. For some, the personal computer came to represent the ultimate tool for giving people access to the power of computer technology. For others it promised the realization of the long-awaited desire to have a computer of one's own.

The first real personal computer was the Altair, introduced in 1974. Unlike earlier computer hardware developments, the Altair was not the result of a multi-million dollar research and development effort. It was the result of a pragmatic attempt to meet a demand among hackers and computer hobbyists for a small, low-cost personal computer. When the Altair was introduced in 1974, the machine aimed for a small, but enthusiastic, market of hackers and electronics hobbyists. The Altair was offered as a kit that had to be carefully assembled, and this in itself limited its market potential. However, what the Altair clearly demonstrated was that one could build small personal computers at a fraction of the cost of other commercially available systems. This signaled the beginning of the personal computer era.

In 1977 several companies, including Apple Computer, Radio Shack, and Commodore, began to offer personal computers. Unlike the Altair, these machines were all complete computer systems that could be put to use without prior assembly. The only problem was that there was hardly any software available for them. For the hackers, this didn't pose much of a problem because

they could easily write their own software. For the vast majority of new computer users, however, the desire for software to run on their newly acquired computers suddenly presented itself as a new and tempting market.

The Killer App

Among the first applications to open up the market for commercial personal computer software were computer games. The success of the early arcade computer games in the 1970s had clearly demonstrated that the home computer market for these types of applications could be substantial, and companies like Atari, Commodore, and others jumped on the bandwagon. Suddenly there was a huge demand for talented and highly skilled programmers such as the hackers. The typical hacker interests and lifestyle was ideally suited for computer games programming, and during this time period the host of new computer game companies absorbed many of them.

As more and more computers found their way into homes and small businesses, it became ever more apparent that software could become the new growth sector in the computer industry. Two people who realized this early on were Bill Gates and Paul Allen. In the 1970s, when the first personal computers became available, Gates was a student at Harvard University, but with all the exciting developments going on it didn't take long before he dropped out to start his own software company called Microsoft in 1975. Microsoft's first product was a version of the BASIC programming language for the Intel microprocessor family used in most personal computers at the time.

Microsoft was not the only actor in the early personal computer software business. In 1979 a small and unknown company named Software Arts Inc. introduced a program called VisiCalc. It would revolutionize the computer

industry in more ways than one. As the first spreadsheet program ever, VisiCalc opened up a whole new application area for personal computers by putting the power of financial planning directly into the hands of ordinary users. What is more important, however, is that VisiCalc also represented the beginning of a change in people's perception of the value of software versus hardware. Since VisiCalc was originally only available on the Apple][computer, many people bought this machine simply to be able to use the VisiCalc program. What's more, VisiCalc offered functionality and features that were not even available on mini computers and mainframe systems at the time, and this really drove home the point that the personal computer was not a toy for hackers, hobbyists, and home gamers, but also a serious business tool. Although VisiCalc was the first software program to achieve such a defining status, other programs would follow. One example is Aldus PageMaker (1985), the program that together with the Apple LaserWriter (1985) created the new area of desktop publishing. PageMaker was made possible by the introduction of Apple Computer's Macintosh model in 1984. The graphical user interface of the Macintosh enabled software makers to explore new application areas such as desktop publishing. Again we see that people began to buy Macintoshes just to be able to use the PageMaker program for desktop publishing. Software was now becoming more important than the hardware on which it ran.

Apple, Microsoft, and the IBM PC

As with most other computer manufacturers in the 1970s, the personal computer took IBM by surprise. Since well before World War II, the company had been far and away the dominant manufacturer of data processing and calculating machinery. Their mainframe systems were tailored to the business markets, and

hardly anyone in blue suits paid any attention to what was happening at the grassroots level. Thus, when the first personal computers, the Altair and subsequently the Apple][, appeared on the scene, IBM did not see them as any threat to their hegemony. The relative quick success of these small and inexpensive systems, however, soon alerted IBM to the fact that future market opportunities might lay in personal computing. For this reason a new division of IBM was established in Boca Raton, Florida to build and manufacture an IBM personal computer.

Apple's success with the Apple][model propelled the little startup company to the forefront of the rapidly growing personal computer industry. The Apple][did not only benefit from the excellent designs of its creator, Stephen Wozniac, but far more important was the fact that hundreds of independent programmers and small software companies provided a large base of software for it. The Apple][was, in effect, the first computer to demonstrate the power of open standards both with regard to hardware extensibility and in regard to software.

When the first IMB PC was introduced in 1981, it was, unlike the IBM's other computer systems, specifically designed to embrace the open standards concept. It was built around an open standards hardware architecture much in the same way the Apple][had been five years earlier, and this allowed third party developers to provide PC users with a host of add-on technologies that IBM alone could not, or would not, have done. More importantly, however, was the fact that its system software was also intended to adhere to open standards. When the PC was created, its designers wanted to get the operating system from an already established vendor. This led IBM to Bill Gates and Microsoft. While Microsoft at the time did not have an operating system that they could sell, Bill Gates immediately realized what a golden opportunity had just knocked on his

door. He promised IBM that he would deliver an operating system for the PC, and with that deal Microsoft embarked on one of the most remarkable business adventures in history—one that would make Bill Gates the richest man in the world.

The deal between Microsoft and IBM over the PC operating system known as MS-DOS (Microsoft Disk Operating System) sealed the fate of the personal computer software industry for many years to come. By controlling the PC's operating system, Microsoft suddenly had both a major source of income to further the company's growth, and complete control over the standards upon which new software was created. The alliance between the IBM PC and Microsoft's MS-DOS operating system software soon became a serious competitor to Apple Computer and its line of Apple][machines. In the early 1980s Apple therefore began development of a new computer that would meet this competition and help the company maintain its role as a market leader. The result of Apple's effort was the Macintosh introduced in 1984. The Macintosh personal computer, with its groundbreaking graphical user interface, was a major achievement and a milestone in the history of modern computing. Still, it failed to make the impact that its creators had hoped for. With the Macintosh, Apple had forgotten its own open standards lesson that had served them so well with the Apple][. The Mac was a closed proprietary system both with regard to its hardware and software. The first Macintosh models did not have any hardware expansion capabilities, and the operating system, despite its elegant and user-friendly concepts, was considered cumbersome and hard to write programs for. Although Apple had quite some success with the Macintosh, the IBM PC and Microsoft had gotten the upper hand.

Steve Jobs presents Macintosh, an "insanely great" computer. 1984.

Due to the PC's open standards, a new generation of computer manufacturers began to appear on the stage in the 1980s. These so-called clone-makers built personal computers that for all intents and purposes were copies of IBM's PC. The IBM-compatible clones soon became IBM's most serious competitors. They could offer personal computers with the same MS-DOS operating system as the IBM PC, thus allowing their machines to run the same software as the PC, and they could sell their machines at a much lower cost. Although the low cost of these clone PCs certainly contributed to their popularity, it was the fact that they

ran Microsoft system software, and therefore could take advantage of the rapidly growing software base for the PC, that mattered to most new computer buyers.

As Microsoft strengthened its grip on the personal computer software market in the 1990s with its Windows operating systems, its only real competitor, Apple, fell further and further behind. The windows systems were close imitations of the Macintosh's graphical user interface, and this, in effect, eliminated the last substantial advantage Apple had over Microsoft. By the end of the decade Microsoft had secured a monopoly on the computer software market—the vast majority of computers made in the world were now Microsoft machines.

The Return of the Hackers

By the mid 1990s, software had made Bill Gates the richest man in the world. His software empire stretched to all corners of the globe, and Microsoft products influenced people's lives in ways that only a handful of technologies have done. Only 20 years earlier few would have predicted that something like this could happen—that software would become such a powerful technological and socially transformative force. The collaborative code-sharing communities of the hackers seemed to have all but disappeared in the face of the new and powerful commercial software industry. Under the shiny surface of commercialism, however, a new generation of hackers was forming a new movement. Linked together via the Internet and the World Wide Web, this was a global movement vastly bigger and more resourceful than prior hacker communities. Many in the new hacker generation were born after the advent of the personal computer, but they were still driven by the same fascination for computer programming that had characterized the first hackers 30 years earlier.

The hacker movement exists, like any other sub-culture, in a state of opposition to a main dominating culture. For the first hackers, the opposition was directed at what they considered the oppressive and exclusionary mainframe culture. For the new generation of hackers, the opposition more and more came to be directed at Microsoft and what was perceived as the company's imperialistic attempts to control and dominate software development with technically inferior products. The hackers had a cause to rally around, now they needed the means to fight back against software imperialism. The project that hackers flocked to was the GNU/Linux operating system. GNU/Linux represented both interesting programming challenges, as well as an alternative to Microsoft's Windows operating systems. Linked together via the Internet, and aided by collaborative tools such as Email, Usenet, Internet Relay Chat, and the World Wide Web, the new global hacker movement of the 1990s embarked upon a formidable project: to create a new Unix-based operating system that could change the world.

The history of the GNU/Linux project and the new hacker movement is covered in more detail in the next two chapters. In what follows I will focus on some of the important software developments that helped build the infrastructure for communication and collaboration among hackers on the Net in the 1980s and 1990s.

Email

By 1990, the digital computer network that we know as the Internet was already a mature infrastructure for worldwide communication and exchange of information. The Unix-based TCP/IP (Transmission-Control Protocol/ Internet Protocol) protocols, first deployed on the ARPANET in 1982, had become the

ubiquitous standard for facilitating traffic across the digital data network, and combined with the growing popularity and affordability of Unix-based systems in the 1980s such as AT&T Unix, BSD, Sun OS and others, this led to a rapid network expansion that accelerated into the 1990s. Although the inventors and designers of the ARPANET had conceived of the network as primarily a system for resource sharing, entrepreneurial users and developers soon found that it could also be an excellent system for messaging and communication of a more social kind. A good number of these communication-oriented network applications that today are the mainstay of electronic communication came about as "unsanctioned hacks." A good case in point is email, arguably the most important and widespread Internet application ever.

In 1971, Ray Tomlinson, an engineer with Massachusetts-based Bolt Beranek and Newman, (a research and development company hired by the United States government's Department of Defense to develop the ARPANET) came up with the idea of using the ARPANET to send electronic mail messages to users in remote locations. The idea of using computers to send electronic messages was not that revolutionary in itself. By the time Tomlinson started working on email in late 1971, users of time-sharing systems had already had the ability to send electronic messages to one another for quite some time. The significance of the new email system, however, was that electronic messages could now be sent across a network of remote computers, thus eliminating geographical barriers in the communication between people. Sending messages to colleagues in the room next door had a certain but somewhat limited usefulness, sending messages to colleagues half a world away and getting immediate responses on the other hand, opened up a whole new set of possibilities. It is interesting to note that email, just like most of the other technologies that I discuss in this study, was

very much a hacker creation. According to Tomlinson himself, he created it "mostly because it seemed like a neat idea… there was no directive to go forth and invent e-mail" (Campbell). Email was, in other words, a neat hack that Tomlinson did because he enjoyed the challenge. This becomes even more apparent when Tomlinson explains how the system was actually implemented. At the time, he had written a simple file-transfer protocol named CPYNET that would act as a bridge to carry data from one computer to another, and this provided the infrastructure that he needed to implement an electronic mail system. He explains:

> I had two programs. One was called SNDMSG, which was used for composing messages to be put in the mailbox of another user in a time-shared computer. There were versions of SNDMSG from Berkeley and MIT, but I recoded it. Another program was an experimental FTP program. I wrote a version to act as a server, another to act as a client, to specify what should be transferred, then send the data of the file to the other computer. I took those two programs and put them together with some glue software, the sticky stuff. Rather than getting the source from a file, you'd get the source from the buffer of the editor. And instead of simply writing the file at the remote end, you would append the characters to the mailbox file. The new message would follow the earlier message that would already be there.
>
> And then there's a thing that everyone remembers, or associates with e-mail, which is the @ sign, which gave the editor a way to specify the recipient. You had to have a way of separating the user and the computer

> name. In English the @ sign is obvious, in other languages it isn't. But being the only preposition on the English keyboard, it just made sense. (Festa)

The development of email, or network mail as it was called at the time, happened to coincide with the design and implementation of the ARPANET file-transfer protocol, and when word about Tomlinson's experimental email system got out, the decision was made to include it for general use on the ARPANET. In this manner, email came to be a standard feature in the early days of the ARPANET, and its success was almost instantaneous. A study conducted in 1973 concluded that at the time "three-quarters of all traffic on the ARPANET was email" (Hafner and Lyon).

The first electronic email-based communities on the ARPANET grew up around mailing lists of various sorts. Since then email systems have been refined and expanded upon numerous times by many programmers, but from the users' point of view little has changed since Tomlinson's original conception of electronic network mail. To this day, email remains a convenient and flexible medium to communicate with colleagues, stay in touch with friends and family, or discuss topics of common interest with strangers elsewhere on the Net.

Usenet

Although email has remained a vital medium of communication among hackers, other media has also been used extensively. The perhaps most popular and influential one in the 1980s and early 1990s was Usenet, also called NetNews or simply News. Like most of the technologies mentioned in this study, Usenet was not created because some government agency or commercial software developer

thought there would be a need for it. It happened simply because a group of dedicated graduate students wanted to make it happen.

The idea that was to become Usenet was born in 1979 by Duke University graduate students Tom Truscott and Jim Ellis. Feeling excluded and left out from the ARPANET, they decided to create their own "General Access Unix Network" with the aim of connecting people with a common interest in Unix. Steven Daniels, also a graduate student at Duke, and who wrote the first C-based News program (A News), explains:

> We (or at least I) had little idea of what was really going on on the ARPANET, but we knew we were excluded. Even if we had been allowed to join, there was no way of coming up with the money. It was commonly accepted at the time that to join the ARPANET took political connections and $100,000. I don't know if that assumption was true, but we were so far from having either connections or $$ that we didn't even try. The `Poor man's ARPANET' was our way of joining the Computer Science community and we made a deliberate attempt to extend it to other not-well-endowed members of the community. (Hauben and Hauben)

With the help of Steve Bellovin, a graduate student from the neighboring University of North Carolina at Chapel Hill, the group soon had a small experimental network running between Duke University (duke), University of North Carolina at Chapel Hill (unc), and the Physiology Department of the Duke Medical School (phs). The News software ran on Unix machines linked together by homemade 300 baud autodialer modems. The system was designed so that an article or news item posted from one of the network nodes would propagate to

other nodes whenever the autodialer opened a connection. In this way, Usenet would automatically synchronize all the information in the network and make it available to users in a timely and organized manner. The first public presentation of Usenet was made in January 1980 at the academic Usenix meeting in Boulder Colorado. The response from the conference participants was overtly positive, and this encouraged the group to go on and create a public release of the Usenet software for general distribution at the 1980 Usenix summer meeting. Documentation accompanying the distribution stated that, "[a] goal of USENET has been to give every UNIX system the opportunity to join and benefit from a computer network (a poor man's ARPANET, if you will)" (Hauben and Hauben). To the surprise of its creators, Usenet grew slowly at first, but when the University of California at Berkeley joined the network by 1981, the expansion rate increased exponentially as links to the ARPANET began to appear in numbers. Between 1979 and 1988 the number of Usenet nodes grew from 3 to more than 11,000, and the newsgroups hosted within were no longer limited to just Unix discussions, they now spanned a vast array of subjects (Hauben and Hauben).

For hackers who might have found themselves as outsiders in their real-world communities, Usenet and BBS (Electronic Bulletin Board) systems that began to appear at the same time (Christensen), provided important escape hatches into new digital communities where they could connect with like-minded individuals who understood their vocations and with whom they could share their interests. In the case of Linux, to be discussed in depth in the next chapter, the news group comp.os.minix, for instance, was an exceedingly important community in the early days as Linus Torvalds strived to conceive and implement his new operating system. Later on, other news groups as well as

mailing lists would fulfill similar functions and help bring more people into the thriving Linux community.

IRC and Other Chat Systems

Whereas email and Usenet proved to be flexible and powerful asynchronous modes of communication, various chat systems became popular media for online synchronous communication. Chat systems in various forms had existed on ARPANET and other networks since the early days, but during the 1980s their popularity and use grew to such an extent that systems administrators became seriously worried about the load they presented on network traffic. In February of 1985, for example, Henry Nussbacher of the Weizmann Institute of Science in Israel conducted a study, concluding that "CHAT rebroadcasters present a very large and growing threat to the BITNET network" (Nussbacher). He went on to recommend that "all people receiving this mail should examine their system for CB CHAT systems and inform the author(s) that running such a system is not allowed." Nussbacher raised some valid concerns since in the BITNET network, messages took priority over file transfers, which meant that in times of high chat volumes file transfers could grind to a halt and thereby obstruct "legitimate" use of the net. Not everyone was willing or able to heed Nussbacher's warning, however. A rapidly growing number of people found online chatting to be not only fun and useful, but also highly addictive, and thus the use of chat systems continued unabated.

One of the best-known chat systems on BITNET in the 1980s was Relay, a program written by Jeff Kell of the University of Tennessee-Chattanooga in 1985. It became a smash hit that propelled online chatting to even greater volumes. As a result, by 1987 Relay was on the verge of becoming a victim of its own success,

not primarily due to the network or CPU load, but because of an increasing number of unruly users who logged in "just to play" and cause problems for the system administrators (Kell).

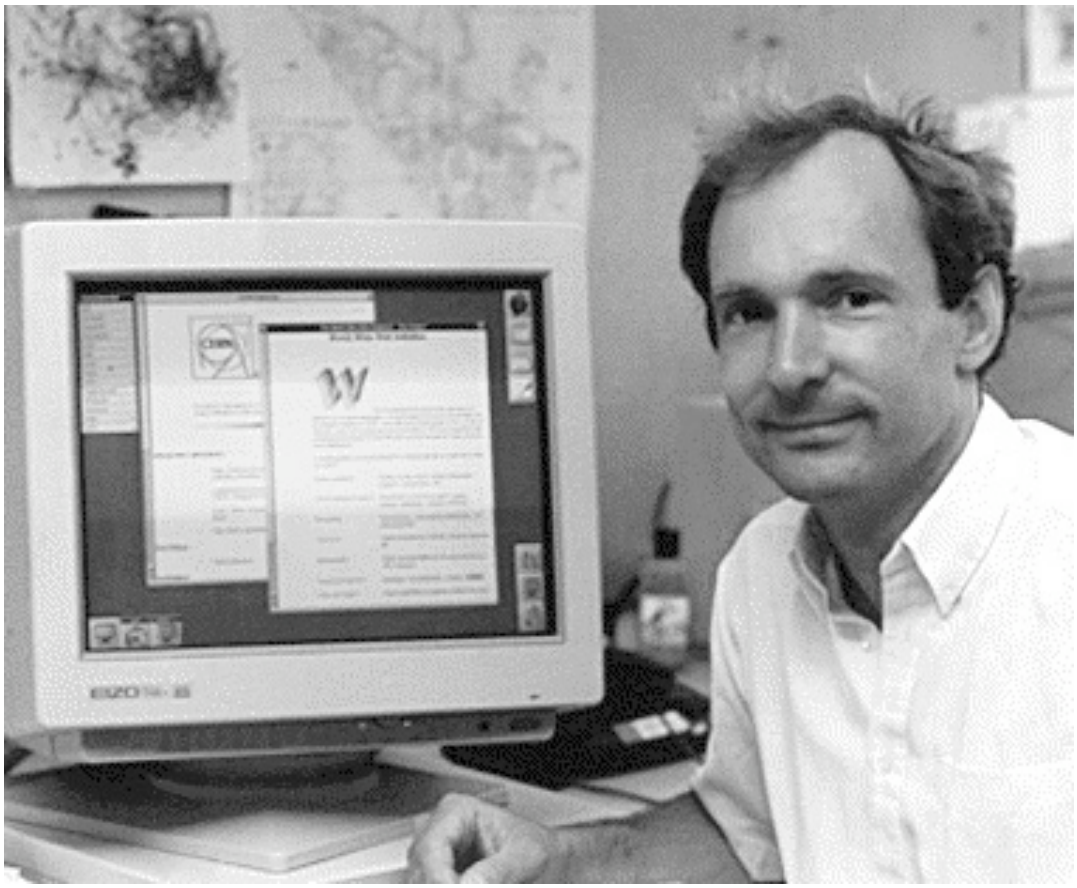
In the 1990s, the most popular chat program was Internet Relay Chat, generally known as IRC. Jarkko Oikarinen, a systems administrator at the University of Oulu in Finland, wrote the original IRC in the summer of 1988. In his job he had a lot of free time on his hands, and part of this time was spent running a public access BBS system (OuluBox) at the university. According to Oikarinen, the ideas behind IRC were inspired by a desire to make the university's BBS more useful by adding "USENET News-kind of discussion and groups […] in addition to real time discussions and other BBS related stuff" (Oikarinen). In the development he also borrowed ideas and concepts from both BITNET Relay Chat, the Unix person-to-person Talk utility. After the first IRC server was up and running on his local machine, Oikarinen sent copies of IRC to friends at other universities in southern Finland, and pretty soon the new chat program had a good size user base among students and academics in Finland. Encouraged by the success at home, Oikarinen says he then "contacted some friends […] through BITNET Relay and asked if they would try this program. Internet connections did not yet work from Finland to other countries, so they could not connect to the Finnish network" (Oikarinen). The recipients of IRC set up their own chat servers and passed along copies to their own friends. In this manner, the IRC program spread quickly, and soon new IRC chat servers were popping up all over the Internet.

IRC first gained international fame during the Persian Gulf War of 1990-91 when hundreds of users tuned into IRC channels such as *#peace* to hear real-time reports from ordinary people in and around the war zone. The news that was

mediated via IRC had a real personal, authentic, and down-to-earth quality to it that in great part contributed to the growing fascination with IRC. In the summer of 1990 there were 38 IRC servers. During the Persian Gulf War, peak usage went up sharply from 100 to 300 simultaneous users. The system continued to grow and spawn new networks such as Undernet, DalNet, EFnet, and others. Another major peak occurred in 1996 when id Software released its highly anticipated first-person shooter game Quake. The IRC channel *#quake* saw more than 1500 users, which made it the largest and most active channel on all the IRC networks. By the end of the 1990s IRC served well over 50,000 users, and although the competition from new chat and instant messaging systems was becoming fiercer, IRC still continued to serve as an important meeting place for hackers and others interested in free and open source software.

The World Wide Web

While the Internet technologies that I have discussed thus far all contributed significantly to the growing popularity of the Internet and its predecessors, no single technology meant more for the astonishing growth and adoption of the Net in the 1990s than the World Wide Web (hereafter referred to as WWW or the web). In the course of just a few years, the Web totally changed the Internet landscape and made it not only attractive and accessible to scores of new users, but also for the first time to commercial interests, business, and industry. The business world's discovery, and subsequent adoption, of the Internet was made possible by deregulations and lifting of restrictions on commercial use in the early 1990s; but it was the Web that would become the real vehicle for the commercialization of the Internet.



Tim Berners-Lee, inventor of the World Wide Web.

In the late 1980s the inventor and principal designer of the World Wide Web, Tim Berners-Lee, was working on ideas for a new information management system for CERN, the European Organization for Nuclear Research. His ultimate goal was to create a system that would constitute a "shared information space through which people (and machines) could communicate" (Berners-Lee "The World Wide Web"). Based on personal experiences with his own information management system, Enquire, written in 1980, and building on the ideas of people like Vannevar Bush (Memex, 1945), Doug Englebart (NSL), and Ted Nelson (Hypertext), in 1989 Berners-Lee came up with a proposal, which in essence became the first conceptualization of the World Wide Web (Berners-Lee "Information Management").

While the first proposal failed to materialize into a concrete project, the same proposal resubmitted two years later met with approval, and in October of 1990 Berners-Lee officially began working on the system that was to become the World Wide Web. In the fall of 1990, Berners-Lee was joined by among others Robert Cailliau, Nicola Pellow, and Bernd Pollermann, and the project picked up pace quickly. By Christmas that year functional prototype versions of both a World Wide Web server and a browser were up and running. The browser had support for the new native Hypertext Transfer Protocol (http) as well as Usenet's nntp protocol. In addition to a graphical browser written for the NeXT operating system, the team had also developed a more limited non-graphical browser for line-mode terminals. Along with the software and protocol specifications, the group also devised a special mark-up language for the Web that they dubbed HTML (HyperText Mark-up Language). The new mark-up language was derived in part from the more complex SGML document markup-system, with the addition of web specific tags and features such as document linking.

In the beginning the only web server available was nxoc01.cern.ch, and the amount of information that it contained was either related to CERN or the Web itself. The Web was a small place indeed back in 1990-91, but WWW group was eager to expand it, and on August 6th, 1991, Berners-Lee made the following seminal announcement to the Usenet news group alt.hypertext:

> The WorldWideWeb (sic) (WWW) project aims to allow links to be made to any information anywhere. […] We have a prototype hypertext editor for the NeXT, and a browser for line mode terminals which runs on almost anything. These can access files either locally, NFS mounted, or via anonymous FTP. They can also go out using a simple protocol (HTTP) to

> a server which interprets some other data and returns equivalent hypertext files. […] If you're interested in using the code, mail me. It's very prototype, but available by anonymous FTP from info.cern.ch. It's copyright CERN but free distribution and use is not normally a problem.
>
> We also have code for a hypertext server. You can use this to make files available (like anonymous FTP but faster because it only uses one connection). You can also hack it to take a hypertext address and generate a virtual hypertext document from any other data you have - database, live data etc. It's just a question of generating plain text or SGML (ugh! but standard) mark-up on the fly. The browsers then parse it on the fly.
>
> The WWW project was started to allow high energy physicists to share data, news, and documentation. We are very interested in spreading the web to other areas, and having gateway servers for other data. Collaborators welcome! (Berners-Lee "Re:Qualifiers")

Coming from an academic background and working in a scientific research institution like CERN, for Berners-Lee, sharing the web source code and concepts was a natural thing to do. In his own words, the principal philosophy behind both the development of the Web and sharing the technology behind it was that "academic information is for all, and it is our duty to make information available" (Berners-Lee "C5-Conclusion"). With the web software freely available via FTP, people soon started to download and experiment with it. Slowly but surely the World Wide Web began to spread, first in Europe, later in the United States, and elsewhere around the world.

Mosaic: Graphical User Interface for the World Wide Web

At the National Center for Supercomputer Applications (NCSA), a research institute at the University of Illinois at Champaign-Urbana, the Web had caught the attention of Joseph Hardin, head of NCSA's Software Development Group. He was intrigued by the possibilities that the web seemed to offer and decided to introduce it to his colleagues. At the time, the Software Development Group had two undergraduate students, Marc Andreessen and Eric Bina, working with them, and under Hardin's direction in early 1993 these two began working on a web browser for the Unix X-Windows system. The program that Andreessen and Bina wrote was Mosaic, widely recognized as one of the most significant contributions to the popularization and growth of the World Wide Web. An alpha version of the Mosaic browser for X-Windows was released on the Internet in February of 1993, with Macintosh and Windows versions following later in the year. In the normal academic tradition, Mosaic was distributed freely from NCSA's FTP servers, and before the year was up a rapidly growing number of people, including the press, were discovering that the Internet had a new feature called the World Wide Web and a new slick interface called Mosaic. There are several reasons why the NCSA Mosaic browser came to have such a profound impact on the future use and direction of the Web. One reason that commentators often point to is the fact that Mosaic became in essence the first graphical user interface to the World Wide Web, and indeed to the Internet as a whole. Contrary to popular belief, however, Mosaic was not the first graphical web browser. Berners-Lee's original WorldWideWeb (sic) browser for the NeXT had a graphical interface and the ability to show graphic web content, and so did other early browsers for the X-Windows system such as the Finnish Erwise

Marc Andreessen. Co-author of the NCSA Mosaic web browser and co-founder of Netscape Communications Corporation.

browser or Pei Wei's Viola browser. Both of these browsers were developed by students and released in the spring of 1992, almost a year before Mosaic became available. Still, Mosaic had one small but incredibly powerful new feature that set it apart from the competition; it was able to display in-line images. The creators of Mosaic had, on their own initiative, added a new HTML tag called IMG that allowed content creators to easily incorporate graphics directly into their web pages. Since Mosaic was the first browser to have this feature, it quickly became the browser of choice for web authors and, by extension, readers as well. Another reason for Mosaic's success must be attributed to its early availability on consumer platforms. While most early browsers, including Berners-Lee's WorldWideWeb, were written for professional Unix systems, Mosaic was readily available for both Mac OS and Windows early on.

Conclusions

In this chapter I have looked at some of the most important developments in the first fifty years of the history of software. Unlike the notion of the special purpose machine born out of the Industrial Revolution, the modern computer as it materialized after World War II was a truly universal machine. Once people realized the significance of this concept, software became one of the most important areas of research and development within the new fields of computer science and engineering. The invention of the compiler and the first high level programming languages in the 1950s made computers more accessible and helped open up commercial markets. Throughout the 1960s much of the scientific research on software was directed toward the development of general-purpose tools and concepts. At the same time, students, who for the first time were able to become involved with computers, developed a whole new hacker culture devoted to programming. The 1970s and 80s were the times when computers became machines for everyone. This was made possible by the invention of the microprocessor and the personal computer, but it was software such as computer games, word processors, spreadsheets, desktop publishing and more which was the true driving force behind it all. In the 1990s the computer morphed into yet another type of device, this time with a focus on communication. The evolution of the Internet and its uses was driven to a significant degree by advances in the software communication technologies associated with it. Most of these technologies were developed by the group of people that I previously referred to as the hackers. From email to Usenet to IRC and the World Wide Web, the hackers stood behind many of the technologies that changed the face of computing in the 1990s. Most importantly, however, the hackers were also largely responsible for the development of the very infrastructure of the Internet,

the Unix-like operating systems that drove it all. The history of this fascinating development is the focus for the next two chapters.

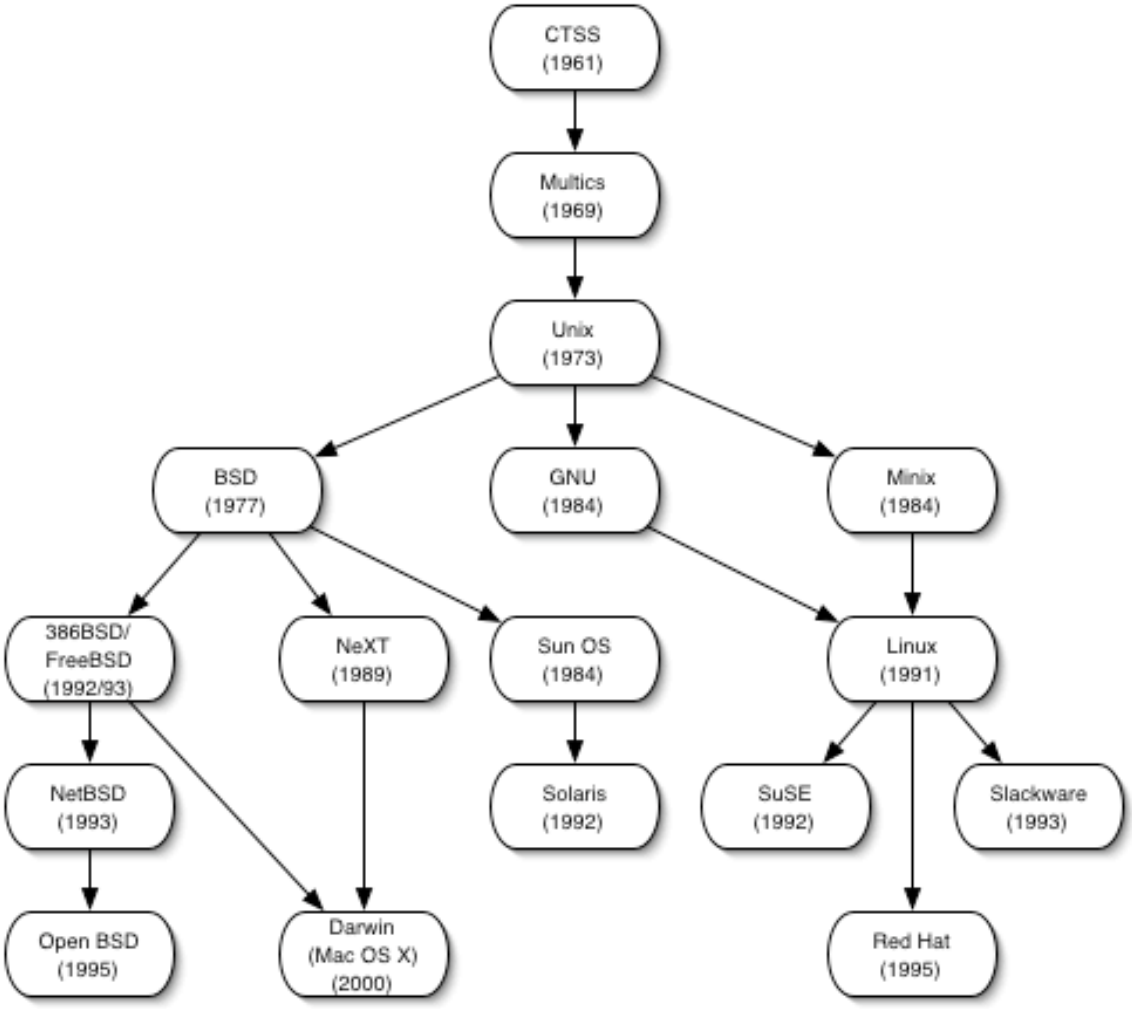
3

Free as in Freedom
The Story of BSD and GNU/Linux

"Free software" is a matter of liberty, not price. To understand the concept, you should think of "free" as in "free speech," not as in "free beer." –Richard Stallman

Operating systems are some of the most complicated pieces of software ever built. As Richard Stallman, leader of the free software movement points out: "With an operating system you can do many things; without one you cannot run a computer at all" (Stallman "The GNU). For this reason, operating systems have continued to occupy the imagination and the creative talent of hackers up until the present day. The following case study looks at the history of the Unix operating system and its derivatives BSD and GNU/Linux. In the context of this study, the history of these systems is significant because it illustrates how hackers have appropriated a technology that was originally developed for scientific research purposes and made it their own.

The chapter opens with an overview of the early history of time-sharing systems. Due to their hands-on character, these systems caught the attention of



Milestones and influences in the evolution of the BSD and GNU/Linux operating systems.

hackers early on, and many of the principal actors in this story cut their first teeth on the experimental systems that were being pioneered at places like the Massachusetts Institute of Technology in the 1960s. The main focus of the chapter, however, is an analysis of the development of the BSD and Linux variants of Unix spanning almost three decades from the mid 1970s until the year 2000. In the course of creating these two systems and related software, the hackers have situated themselves as a community with shared interests, values, ethics, and goals that are manifested through the philosophy of free software and

the concept of open source. Through their appropriation and adaptation of Unix technology, and with the help of the Internet, the hackers have also forged new collaborative models of development that by the turn of the century promised to change the face of software engineering.

Time-Sharing and the Roots of Unix

During the 1950s, advances in digital computing not only produced faster and more versatile hardware, it also spawned a new area of research centered on the development of software solutions ranging from operating systems to programming languages. Still in its infancy, digital computing posed an endless array of interesting challenges both of a practical engineering nature, and of a more theoretical and intellectual nature. One of the challenges that rose to prominence around 1960 was the question of how computers could be used in ways that took better advantage of both the technical advances that had been achieved, and that was better suited to face new problems and challenges in the emerging area of software engineering.

In the 1940s and 50s, the predominant mode of using computers was through a method known as batch-processing. This was a scheme that had been used with punch-card data processing machinery since the late 19th century, and meant in essence that users interacted with the machine in an asynchronous fashion. A user had to prepare her data, for example a computer program, in advance, then take it to the computer were it was placed in a queue. She then had to wait to obtain the results until the computer had worked its way through all prior jobs in the queue. This could take hours or even days. For a programmer, batch processing posed several significant problems. For example, if the computer encountered a bug while running a program, it would stop and return an error

message. The programmer would then have to take the program back to her office, fix the problem and return to the computer and run it through the batch system again and again until the program finally ran the way it was supposed to.

Although batch processing was the order of the day in the 1950s, alternative models did exist, if only in very experimental ways. Between 1946 and 1951, a group of engineers and scientists working under the direction of Jay Forrester at MIT had developed "Project Whirlwind," a digital computer using a groundbreaking random-access magnetic-core memory technology that Forrester had invented. Whirlwind was originally intended to be a general-purpose flight simulator for the United States Navy, but over time it evolved into what became, in essence, the world's first real-time general purpose computer (cf. Redmond and Smith). By 1960 technical advances such as the transistor-based TX-0 computer developed at MIT's Lincoln Lab in late 1957, in combination with CRT-type monitors, further demonstrated the promise of real-time digital computing. With the technical opportunities at hand, it didn't take long for the MIT research scientists to start thinking about ways to implement software solutions to make real-time computing a reality. What took place during this process was the beginning of a new paradigm in the way people use computers. That paradigm would be known as time-sharing.

John McCarthy, one of the pioneers behind the concept of time-sharing, explains that "by time-sharing, I meant an operating system that permits each user of a computer to behave as though he were in sole control of a computer, not necessarily identical with the machine on which the operating system is running" (McCarthy "Reminiscences"). McCarthy had first started thinking about time-sharing as early as the fall of 1957, when he first arrived at MIT's Computation Center in Cambridge, Massachusetts. In a memo from January of 1959, McCarthy

outlines his thoughts on the benefits and implications of such a system. He begins by saying that the goal is to develop "an operating system for it [IBM 709] that will substantially reduce the time required to get a problem solved on the machine." He goes on to say that "I think the proposal points to the way all computers will be operated in the future, and we have a chance to pioneer a big step forward in the way computers are used" (McCarthy "A Time-Sharing").

McCarthy's ideas were met with enthusiasm among his colleagues at MIT, and over the next couple of years the research scientists there worked on various ways to implement a system in which users could share the use of one computer. Much of this early work involved hardware modifications to IBM-machines on which the time-sharing system was implemented, something that required a close collaboration with IBM. The first tangible result of the efforts to build a time-sharing operating system was the Compatible Timesharing System, CTSS for short, and so named because it had to be compatible with existing batch-processing systems that were also in use at the time. CTSS was developed under the direction of Fernando J. Corbató of MIT's Computation Center and first unveiled in the fall of 1961. By 1965, CTSS had been implemented to run on a modified IBM 7094. It could support 30 simultaneous users, and clearly demonstrated that computers could be utilized much more efficiently and creatively when users were allowed to work with the machine in real time.

Due largely to the success of CTSS and the promise of time-sharing, in 1963 MIT established Project MAC (Multiple Access Computers/Man and Computer), whose main focus was to further the research and development in this area. A significant portion of the funding came from the U.S. Department of Defense's Advanced Research Projects Agency (ARPA). The project had two prongs, one of which was the development of a new time-sharing operating system dubbed

Multics (Multiplexed Information and Computing Service) led by Corbató. Multics was an ambitious research project where "one of the overall design goals is to create a computing system which is capable of meeting almost all of the present and near-future requirements of a large computer utility" (Corbató and Vyssotsky). It was designed to be a true time-sharing system, and in order to be as platform independent as possible, the whole operating system was to be implemented in a new high-level programming language from IBM called PL/1. Other cutting-edge features included virtual memory and support for multiple processors. In 1964 the MAC team chose a GE-645 machine from General Electric (GE) for the first implementation of Multics. A year later, Bell Labs decided to acquire the same type of machine and subsequently became involved in the Multics project with MIT and GE. Although the project now had significant resources at hand, the development did not progress as quickly or as smoothly as one had hoped. The choice of PL/1 as the implementation language slowed down the progress, partly because it was harder than expected to implement it on the GE-645, and partly because it took a lot longer than anticipated to produce a decent compiler for it. Several times the Multics project came close to cancellation, and in April of 1969 Bell Labs decided to withdraw from the project altogether. A few months later, in October of that year, an early version of Multics finally became operational ("Multics History"). Although Multics never became a widely used system, as a research project it was an important stepping-stone in operating system design. Many of the key features and concepts found in later systems were invented and first appeared in Multics, and several of the project members later went on to become key figures in computer science. Two of those people were Bell Labs researchers Ken Thompson and Dennis Ritchie.

Unix

After Bell Labs withdrew from the Multics project, Thompson, Ritchie, and a few others began searching for ways to create an alternative to the Multics operating system. Ritchie explains:

> What we wanted to preserve was not just a good environment in which to do programming, but a system around which a fellowship could form. We knew from experience that the essence of communal computing, as supplied by remote-access, time-shared machines, is not just to type programs into a terminal instead of a keypunch, but to encourage close communication. (Ritchie "The Evolution")

Throughout 1969, Thompson in particular spent a good deal of time experimenting with design and implementation of core components of a new operating system. These included a hierarchical, Multics-like file system, an assembler for implementation of programs, a command interpreter (shell) and a small set of utilities such as copy, print, and delete for easy file manipulation. During this time, Thompson also wrote a game named *Space Travel* (a simulation of the movement of stars and planets in the solar system) that served as "an introduction to the clumsy technology of preparing programs for the PDP-7" (Ritchie "The Evolution"). With these basics components in place, Thompson recalls that he then "picked up a couple of users, Doug McIlroy and Dennis Ritchie, who were interested in languages. Their criticism, which was very expert and very harsh, led to a couple of rewrites in PDP-7 assembly" (Cooke et al). A new operating system was taking shape, but according to Ritchie, it was "not until well into 1970 that Brian Kernighan suggested the name 'Unix,' in a

Ken Thompson (left) and Dennis Ritchie in the 1970s

somewhat treacherous pun on 'Multics,' [that] the operating system we know today was born" (Ritchie "The Evolution").

The earliest versions of the Unix operating system (V1-V3) were written in assembler. This ensured that the system was fast and responsive, but it was not particularly portable since its code was highly machine-specific. Another drawback of the assembly language used in the early implementations of Unix, according to Ritchie, was that it did not have any mechanisms such as loader and link-editor, which meant, among other things, that one could not make use of libraries. Every program had to be complete in itself, and this led to a significant code redundancy (Ritchie "The Evolution"). In an effort to surmount these problems Thompson first attempted to re-implement the system in a language called BCPL. He says, "I thought [it] was a fairly straight translation, but it turned out to be a different language so I called it B, and then Dennis took it and added types and called it C" (Cooke et al). With the advent of C, Thompson and Ritchie had a development tool that supported both high-level portability and modularization and low-level efficiency through the incorporation of assembly

code. Thompson says, "we tried to rewrite Unix in this higher-level language that was evolving simultaneously. It's hard to say who was pushing whom—whether Unix was pushing C or C was pushing Unix" (Cooke et al). The first C implementation of Unix was released as version V4 in 1973.

Although Unix was developed in the setting of a traditional research laboratory, namely AT&T's Bell Labs, its creation had many similarities with that of a hacker system. According to Ritchie, "it was never a 'project'; it was not designed to meet any specific need except that felt by its major author, Ken Thompson, and soon after its origin by the author of this paper, for a pleasant environment in which to write and use programs" (Ritchie "Retrospective"). Perhaps for this very reason Unix quickly became the system of choice for hackers, programmers, and system administrators alike. It was open and minimalist, yet powerful and flexible. It allowed its users complete access to the inner workings, but it also had a rigorous permission system in place that ensured the necessary stability needed in a multi-user environment. In combination with Dennis Ritchie's C programming language, Unix became an ideal environment for program development, but perhaps more important, it became one of the most portable operating systems ever made. By the year 2000 no other operating system ran on more platforms than Unix and its derivatives. In the remainder of this chapter I focus on two of the systems that came about because of Unix: BSD and Linux.

BSD: The Berkeley Software Distribution

The origins of the Berkeley Software Distribution, commonly known as BSD, can be traced back to the Fall of 1973, when Robert Fabry, a professor at the University of California at Berkeley, learned about Unix from a conference

presentation by Thompson and Ritchie. This was the first public presentation of Unix, and Fabry became so interested that he approached the developers to obtain a copy (McKusick). Berkeley was, at the time, in the process of acquiring a new PDP-11 mini computer, and Fabry felt that Unix would be a perfect match for that machine. His reasoning for wanting Unix was quite pragmatic. Due to anti-trust restrictions imposed on AT&T by the U.S. government, in which the company was not allowed to benefit commercially from non telephony-related inventions, AT&T had to sell Unix at a very low cost. Furthermore, for an additional $99, academic and government institutions could obtain a license to the system's source code. Compared to the competition, Unix was also a highly cost efficient system to run and maintain. Whereas the cost per user on a typical mainframe multi-user system could easily reach $50,000, the cost per user on a mini-system running Unix could be as little as $5,000 (Leonard "BSD Unix").

After Unix version V4 was installed at Berkeley in 1974, it quickly began to outpace the existing batch processing system in terms of popularity, especially among the students. It was not, however, until Thompson, himself a graduate of the University of California at Berkeley, arrived to spend the 1975-76 academic year as a visiting professor in the Computer Science department that a more genuine interest in the inner workings of the system arose. During his tenure at Berkeley, Thompson worked on a revision of the system dubbed V6, and perhaps more importantly, he taught Unix. For many, learning Unix directly from the master was a revelation. Fabry recalls, "we all sat around in Cory Hall and Ken Thompson read code with us. We went through the kernel line by line in a series of evening meetings; he just explained what everything did. […] It was wonderful" (Leonard "BSD Unix"). Not surprisingly, students in particular became fascinated with Unix through Thompson's teachings. One of those

students was Bill Joy, who had just started his undergraduate studies at Berkeley in the fall of 1975.

Thompson's presence at Berkeley in the mid 1970s was an important catalyst for the BSD development that later ensued. On the one hand, he helped create an intellectual environment in which to study and learn the Unix system; on the other hand, as an accomplished programmer and hacker he became a model and a source of inspiration for aspiring young programmers like Joy and others. A good example of Thompson's practical influence at Berkeley was a Pascal system that he "had hacked together while hanging around the [PDP] 11/70 machine room" (McKusick). As with any other hack, the system had plenty of room for improvements, something that Bill Joy soon discovered. Joy was at the time involved in a student-programming project using Thompson's Pascal system. He explains:

> I tried to write the thing in Pascal because Pascal had sets, which Ken Thompson had permitted to be of arbitrary length. The program worked, but it was almost 200 lines long - almost too big for the Pascal system. I talked to Thompson to figure out how I could make the Pascal system handle this program. Chuck Haley and I got involved in trying to make the Pascal system handle it, but the thing was wrong because it was building the entire thing in core. So I got sucked in, got department help, and built some hope of receiving enough support eventually to pay for this program to work under Pascal. (Joyce "Interview")

According to McKusick, over the course of the year, Joy and Haley "expanded and improved the Pascal interpreter to the point that it became the programming

system of choice for students because of its excellent error recovery scheme and fast compile and execute time" (McKusick). Chuck Haley earned his Ph.D. for his work on the revised Pascal system and left Berkeley. For Bill Joy, however, it was only the beginning.

After Thompson returned to Bell Labs in 1976, Joy began to take more and more interest in the Unix kernel. He had all the source code available, and in the true hacker spirit soon began to make little improvements, additions and enhancements here and there. By this time word of the improved Pascal system had gotten out and requests for it started to come in. It was therefore decided to put together a distribution of Unix containing the Berkeley enhancements. The distribution became known as the Berkeley Software Distribution (BSD) and was built and released by Bill Joy in early 1977.

"Read the Protocol and Write the Code"

In the context of this study, the history of BSD is particularly interesting because of the way it illustrates a development model for, and the evolution of, a large collaborative software project. The basis for BSD was, as I have mentioned, the Unix source code written mostly by Ken Thompson. As such, BSD was not, at least in the beginning, an operating system in itself. It was, as the name implies, a distribution of Unix that included certain additions and enhancements developed by the hackers at Berkeley. In the early days, the most notable additions were Joy and Haley's improved Pascal system, and an early version of a small editor that later morphed into a Unix mainstay tool for text manipulation—vi (McKusick). The Berkeley hackers' motivation for creating BSD was, in other words, not part of a scheme to replace Unix, or even create a new operating system. By experimenting and playing with the code, they had come up with certain

Bill Joy. BSD hacker and co-founder of Sun Microsystems.

enhancements that they thought were clever and useful. When others in turn came to the same conclusion, sharing the fruits of their work was simply the natural thing to do.

Over the next few years, Joy produced several new releases of BSD. With each new release new features and enhancements were added, so that over time BSD came to have its own distinct flavor. In hindsight, perhaps the most significant of these additions was the inclusion of the TCP/IP (Transmission Control Protocol/ Internet Protocol) protocols, which appeared in 4.2BSD in 1983. TCP/IP networking was added to BSD at the behest of The Advanced Research Projects Agency (ARPA), which was interested in using the system as a standard for computers on the emerging ARPANET, precursor of the Internet. ARPA's reasoning was that it would be much more practical to standardize the network on software rather than hardware, and BSD, being an essentially free academic product, fit the bill perfectly (McKusick). On Fabry's initiative, in 1980 a working group named Computer Systems Research Group (CSRG) was established to develop an enhanced version of the then current 3BSD for the ARPA community.

Fabry hired Bill Joy as the project leader, and with the backing of ARPA and the organizational framework of the CSRG, BSD's popularity increased steadily.

Although every copy of BSD was shipped at a nominal cost complete with source code, contributions and feedback from the user community was, according to Joy, sparse at first (Leonard "BSD Unix"). He held code destined for inclusion in the distribution to a very high standard, and he maintained strict control with every aspect of the project, and this may well have discouraged people from contributing. Among his colleagues and peers he earned a reputation for being both an arrogant and a brilliant hacker. Marshall McKusick says:

> Bill's very good at taking something, [...] saying, 'OK, this is what I have, this is where I want to get to, what's the shortest path from here to there?' His code was ugly, unmaintainable, incomprehensible, but by golly it only took him two weeks to do an incredible amount of functionality. (Leonard "BSD Unix")

Arrogance and brilliance are character traits that almost every true hacker possesses. Bill Joy's many and substantial contributions to BSD in the late 1970s and early 1980s elevated him to stardom within the hacker community. Without knowing it, he came to represent the kind of personality that has colored many people's perception of BSD developers ever since. When once asked how he had accomplished some particularly clever piece of coding, he remarked in the proverbial hacker way: "Read the protocol and write the code" (Leonard "BSD Unix").

BSD, A Model for Collaborative Software Development

In the spring of 1982, Bill Joy left Berkeley to co-found a new company, Sun Microsystems, and the BSD project was eventually taken over by Marshall Kirk McKusick, another Berkeley graduate student. McKusick was much more pragmatic and open to input from the user community, and during his tenure BSD evolved into what was essentially the first large-scale example of what people would later call an open source project. He says:

> The contribution that we made was in developing a model for doing open-source software ... We figured out how you could take a small group of people and coordinate a software project where you have several hundred people working on it. (Leonard "BSD Unix")

The collaborative model that the BSD group pioneered was one of layers. At the center of the project sat a core group of people whose job it was to oversee the project's overall goals and directions. The middle layer consisted of trusted contributors who had access to the official source code repository, and who had permission to commit changes to that repository. The outer layer consisted of programmers who only had access to read code in the source repository. These people could submit bug reports and suggestions for improvements, but they did not have the privilege to make changes to the source code directly.

BSD and AT&T Unix had coexisted peacefully and shared a mutually beneficial research and development environment from the very beginning, but when the AT&T monopoly was broken up in 1984, and the company began efforts to commercialize Unix, the ties between the two began to weaken. In 1992 when the University of California, along with BSDi, a spin-off company from its

Computer Systems Research Group, began selling a commercial version of BSD, AT&T sued over copyright infringements. What AT&T had not taken into account, however, was that due to the long and close collaboration with the BSD group, their own commercial Unix system also contained massive amounts of code from BSD, most notably the BSD TCP/IP stack. The University of California promptly counter-sued AT&T for the same copyright violations. After a prolonged court battle, the case was finally settled in 1994. BSD was allowed to continue their distribution, but they had to remove certain files and could no longer use the trademark Unix. A new release called 4.4BSD was promptly put together, and under the terms of the lawsuit settlement, anyone using this as a basis for new BSD-based distributions was out of harm's way legally.

BSD on the PC

Perhaps the single most important event in the computing world in the 1980s was the coming of age of the personal computer. By the end of the decade these affordable machines had reached a level of power and sophistication that promised to give mini-systems such as DEC's VAX series, or Sun's professional workstations, serious competition. Incidentally, the first effort to create a Unix-like operating system for personal computers was made by Microsoft as early as 1980. The system Microsoft developed, called XENIX, was based on AT&T's Unix System V and was intended for 16-bit microprocessors. When the company, a year later, eventually decided to develop their own operating system called MS-DOS for the IBM PC, the XENIX effort was stranded until 1983 when a new company known as the Santa Cruz Operation (SCO) entered the stage with a system called SCO XENIX for Intel 8086 and 8088 based PCs (SCO "History of SCO"). While SCO in the 1980s established itself as a big Unix player in the

microcomputer arena, the BSD camp was firmly planted in the world of mini machines. By 1990, however, more and more people were lamenting the fact that there was no BSD available for PC. One of those was BSD developer Bill Jolitz. He felt that the BSD porting efforts were not keeping up with the times, and together with his wife, Lynne, Jolitz proposed to undertake the job of porting BSD to the PC. The initial release of the system, dubbed 386BSD, happened on March 17, 1991 (Chalmers). It was a bootable, but very rudimentary BSD system, and there were many outstanding issues that had to be fixed and worked out before it would be truly useful. Over the next year, the Jolitzs worked hard on a more feature-complete release, and on July 14, 1992, they could proudly announce:

> We are pleased to announce the official release of 386BSD Release 0.1, the second edition of the 386BSD operating system created and developed by William and Lynne Jolitz and enhanced further with novel work and contributions from the dedicated 386BSD User Community. Like its predecessor, 386BSD Release 0.0, Release 0.1 comprises an entire and complete UNIX-like operating system for the 80386/80486-based AT Personal Computer. (Chalmers)

Since 386BSD was released entirely on the Internet via FTP, according to McKusick, "within weeks [it] had a huge following" (McKusick). The Jolitzes had expected a few hundred downloads of the system, but in all there were more than 250,000 (Chalmers). Although the Jolitz's BSD port was both well designed and functional, they did not manage to keep up with the inevitable flood of bug reports and other contributions garnered from the ever-expanding user

community. They also had their day jobs to consider, and for this reason, work on 386BSD revision did not proceed as fast as many users wanted. When the much-revised version 1.0 finally appeared in December of 1993, much of the initial momentum had been lost. In the time that had passed between the initial and the final releases, the idea of Unix on the PC had spread like wildfire across the Internet and several more players emerged in the arena. Three of these came from the BSD world.

The first one, NetBSD, was formed in early 1993 by a group of avid 386BSD users who were particularly interested in system portability and multi-platform support. Their vision was to make a BSD distribution that would run on as many computer platforms as possible. Between 1993 and 2001, the NetBSD project released 16 versions of their system, and at the time of this writing (2003), the system runs on over 45 different platforms ("NetBSD"). In the mid 1990s, the NetBSD project split in two with the formation of a new group calling themselves OpenBSD. The principal aim of OpenBSD was to improve the security of the BSD system, and at the same time make it easier to use for inexperienced system administrators. The third, and perhaps most influential, spin off from 386BSD was called FreeBSD. The project was started by Jordan Hubbard, and others, in 1993 in response to what they felt were the Jolitz's reluctance and inability to handle and incorporate feedback from 386BSD users (Hubbard). When the Jolitzes refused to accept their contributions, the FreeBSD group simply decided to build their own distribution, the first of which was released in December of 1993. Although the various BSD-based distributions for the PC enjoyed a reasonable amount of popularity in the 1990s, they totally paled in comparison with another and quite unexpected challenger, Linux. The remainder of this chapter is devoted to the history of the Linux operating system and how it took

the PC Unix market by storm in the 1990s and gave a whole generation a new perception of how to develop, market and distribute software.

Between 1977 and 2000, BSD evolved from a one-man hack to a Berkeley-centered cooperative effort to a distributed collaborative endeavor involving thousands of hackers and programmers all over the world. Along the way, it contributed significant portions of code and technical design features to almost every other Unix-like operating system in existence, helped foster the Internet, and pioneered a collaborative open source model of software development. The historical significance of BSD runs across all these dimensions, and yet it all comes down to one thing. In his essay, "Twenty Years of Berkeley Unix," longtime BSD hacker and distribution maintainer Marshall McKusick sums it up with the following words:

> The history of the Unix system and the BSD system in particular had shown the power of making the source available to the users. Instead of passively using the system, they actively worked to fix bugs, improve performance and functionality, and even add completely new features. (McKusick)

GNU's not Unix

The single most important reason for the resurgence of the collaborative software development model in the 1990s was the creation and evolution of the Linux operating system. It represents the hacker movement's perhaps greatest achievement, not in technical terms, but because it captured the imagination of a whole new generation of hackers, computer users and journalists alike, and brought the notion of collaborative development back into the mainstream of

software engineering. In order to adequately understand the story of Linux, we need to go back to the mid 1980s and examine the developments that made it all possible.

On September 27, 1983, Richard Stallman, a long-time hacker of MIT's Artificial Intelligence Lab, posted the following announcement to the Usenet newsgroups net.unix-wizards and net.usoft (Stallman "Initial Announcement"):

> From CSvax:pur-ee:inuxc!ixn5c!ihnp4!houxm!mhuxi!eagle!mit-vax!mit-eddie!RMS@MIT-OZ
> From: RMS%MIT-OZ@mit-eddie
> Newsgroups: net.unix-wizards,net.usoft
> Subject: new UNIX implementation
> Date: Tue, 27-Sep-83 12:35:59 EST
> Organization: MIT AI Lab, Cambridge, MA
>
> Free Unix!
>
> Starting this Thanksgiving I am going to write a complete Unix-compatible software system called GNU (for Gnu's Not Unix), and give it away free to everyone who can use it. Contributions of time, money, programs and equipment are greatly needed.
>
> To begin with, GNU will be a kernel plus all the utilities needed to write and run C programs: editor, shell, C compiler, linker, assembler, and a few other things. After this we will add a text formatter, a YACC, an Empire game, a spreadsheet, and hundreds of other things. We hope to supply,

> eventually, everything useful that normally comes with a Unix system, and anything else useful, including on-line and hardcopy documentation.

Having been raised in the academic code-sharing environment of MIT, Stallman had come to regard the ongoing commercialization of software with great skepticism and distrust. As he saw it, the traditional gift-sharing economy of software engineering was gradually being replaced by a commercial proprietary model designed to take away programmers' access to code and the ability to freely share computer programs. Prompted by MIT's decision in 1982 to move from their own ITS (Incompatible Time-sharing System) operating system to DEC's new proprietary VAX system, Stallman decided it was time to do something to stem this trend (Stallman "The GNU"). He says:

> The answer was clear: what was needed first was an operating system. That is the crucial software for starting to use a computer. With an operating system, you can do many things; without one, you cannot run the computer at all. With a free operating system, we could again have a community of cooperating hackers—and invite anyone to join. And anyone would be able to use a computer without starting out by conspiring to deprive his or her friends. (Stallman "The GNU")

In January of 1984, Stallman quit his job at the Artificial Intelligence Lab to begin the GNU project. He felt that this was a necessary move in order to ensure that MIT could not lay claim to, or place restrictions on, GNU. The name GNU, says Stallman, "was chosen following a hacker tradition, as a recursive acronym for 'GNU's Not Unix' " (Stallman "The GNU").

Richard Stallman. Photo courtesy of Sam Ogden.

In his Usenet announcement, Stallman said that GNU, to begin with, would consist of a "kernel plus all the utilities needed to write and run C programs." Being the core of the operating system, the kernel might have seemed like an obvious place to start; but as it turned out, it would, in fact, be the last piece of the puzzle to fall into place. Instead, Stallman ended up starting at the opposite end by writing an editor called GNU Emacs that could be used to implement the other pieces of the system (Chassell "Interview"). This editor was based on a macro collection known as Emacs (Editor MACroS) that Stallman had written in 1976 for the ITS editor TECO (GNU "Emacs FAQ"). In view of the future prospects for GNU, Stallman's decision to start off with Emacs instead of the

kernel was shrewd because it meant that the project could deliver something useful almost right away. Whether Stallman was consciously aware of this at the time, or whether it was the result of lucky circumstances is not entirely clear. What seems quite clear, however, is that had he followed his original intentions as outlined in his Usenet posting, the GNU project might never have gotten off the ground. A kernel by itself has almost no practical use. A fully featured, state-of-the-art editor that could be ported and used freely on any type of Unix, and Unix-like system, however, was a different proposition altogether.

"Copyleft—all rights reversed"

The first version of GNU Emacs was put up for distribution via FTP in early 1985. Since many people did not have Net access at that time, and because Stallman was trying to make a living from writing and distributing free software, he also set up a small distribution business that sold the program on tape for $150 a copy. GNU Emacs became an almost instant hit with users, and soon the tape distribution business became too much for Stallman to handle alone. The success of GNU Emacs also brought more volunteers to the GNU project, and it was therefore decided to form a tax-exempt charity organization that would act as an institutional framework for GNU and its various activities. The Free Software Foundation (FSF), as the organization was called, was incorporated in October of 1985 with Robert Chassell as founding director and treasurer. Stallman's aim with GNU was to create a free alternative to Unix that he could "give away to anyone" (Stallman "Initial Announcement"). From this and other early references it is clear that he wanted the software to be *gratis*. Stallman has since tried to downplay this fact, something that I will return to in a later chapter. More importantly, however, he also wanted GNU to be free in the sense that anyone

could use, copy, modify and redistribute it. Chassell explains: "Richard set the original goals, which was to create a system that wasn't even necessarily as good as Unix but somewhat like Unix with the condition only that it be free and not restricted" (Chassell "Interview").

To achieve this goal, GNU needed a license, a binding legal document that would set forth the terms and conditions for using the software. The license that the SFS eventually came up with was called the GNU General Public License (GPL), and it would become perhaps the most important document ever written by the Free Software movement. In the GPL, users of GNU software are guaranteed the rights to use, copy, modify and distribute, and programmers who write the software are protected from companies or individuals who may want to use their code in proprietary software. If anyone makes a modification and redistribution of software under the GPL, they are required by the license to give users of the derivative software the same rights they originally had. This concept was dubbed copyleft, and it was designed to prevent free software from ever becoming proprietary and closed source. Furthermore, the GPL became an important instrument in fostering the self-sustained growth of the free software community since programmers were required to share their modifications with the community from which they got it in the first place.

Building an Operating System Bit by Bit

"The easiest way to develop [the] components of GNU," says Stallman, "was to do it on a Unix system, and replace the components of that system one by one" (Stallman "The GNU"). Thus, after he had finished the GNU Emacs editing environment, he immediately cast his eyes upon what he identified as the next major component, the developer tools. These were the tools needed to actually

implement the remaining parts of the system, and included most notably a compiler, a debugger, and a set of libraries. Early in the project, Stallman had made some attempts to get permission to use certain existing compiler tools for GNU, but when these efforts eventually stranded, he decided to go ahead and write the compiler himself. C was the language of choice on the Unix platform, so a C compiler was what he wrote. The GNU C Compiler, popularly known as GCC, was first released in beta form in March of 1987, with the following official 1.0 release on May 23 that year. More than perhaps any other component of the GNU system, GCC has been the major driving force in the creation of free and open source software. Over the years, it has been ported to nearly every Unix system in existence, and thus, it has provided programmers with a familiar, multi-platform programming environment that not only saved them thousands of dollars in licensing fees, but also allowed them to make modifications and additions for their own particular needs and challenges. Today GCC is known as the GNU Compiler Collection, and in addition to Stallman's original C compiler, it now also contains front-ends for several other popular programming languages such as C++, Objective C, Fortran, and Java ("GCC").

Although Richard Stallman was exceedingly important in jumpstarting and driving the GNU project in the mid 1980s, a growing number of other contributors also joined the ranks. While many of these people made valuable contributions to the project, FSF had a hard time getting volunteers to take on development of major remaining GNU components. Robert Chassell explains:

> We had lots and lots of volunteers. I think at that time the number of people volunteering to some extent or another may have been a few hundred. It's hard to estimate because when you think in terms of how

> these people got involved, well, if someone sends a patch to Emacs which lots of people did, I think of them as a volunteer, but what we were concerned with, what was more important over the next seven or eight years, was getting things done that volunteers wouldn't do. (Chassell)

An example of a component that the volunteers, according to Chassell, were reluctant to take on was the GNU C library, a set of common functions and procedures that defines the system calls and other basic facilities. This was a major undertaking that the FSF felt had to be approached, designed and implemented in a systematic and thorough way in order to produce a truly useful resource that would stand the test of time. For this reason, they decided that the best course of action would be to hire a programmer, Roland McGrath, rather than be dependent on a volunteer effort. Nevertheless, contributions from volunteer programmers continued to be important for the GNU project throughout the 1980s. Chassell says:

> People did develop on their own especially things like little utilities because they aren't huge projects, or people could add to a project that was well started like Emacs and GCC, but they wouldn't start it on their own because it was too big. So part of the psychology of all this is that it's easy to have people start and work on and even complete small projects, but big projects are more difficult. (Chassell)

By 1990 the GNU system was almost complete to the point where it could be used by itself. The only major missing part was the kernel, the core of the system upon which all the other components would run. At the time, Carnegie Mellon

University had developed a promising and free microkernel called Mach, and it was decided to implement the GNU kernel, as a collection of servers running on top of Mach to implement file systems, network protocols, file access control, and other vital operating system features. In the finest hacker naming tradition, the kernel was named HURD which in reality is a pair of mutually recursive acronyms, where on the first recursive level, the acronym HURD stands for "HIRD of Unix-Replacing Daemons," and on the second level, the acronym HIRD, stands for "HURD of Interfaces Representing Depth." If we look at the recursive nature of the name, we'll see that what it actually represents is an infinite loop. Incidentally, that is also what came to characterize the whole GNU HURD project. From the start, HURD was a highly ambitious project that aimed to create the best and most advanced kernel available. Needless to say, accomplishing such a feat would not be done in a day, and for this reason, the project dragged on and on throughout the 1990s. Chassell explains:

> I think that Richard and others got seduced by the promise of the GNU HURD, and it went directly contrary to what he had said in various policy statements earlier, which was that anything used by a free software GNU system needs only to be almost as good as Unix. It just needs to be a suitable replacement. But the theory behind the HURD was that it would actually be better, and there is a major driving force in people writing code to write stuff that's better. It's a real personal motivator, and I think part of what happened with the GNU HURD is that people got motivated to do something that was really better and as far as I can figure out, the GNU HURD in theory is at least still better than any contemporary operating system. (Chassell)

When HURD was first presented publicly at a conference organized by the FSF in 1996, it was immediately met with skepticism among the hackers. Eric Raymond, who was also at the conference, remembers talking to Keith Bostic, a fellow hacker from the BSD community, during a conference break:

> Keith gives me this sort of troubled look and says, "So what do you think?" I looked at him and I said what he was thinking. I said, "it's beautiful, it's elegant, it's elaborate, it's ornate, it's huge and it's going to be killed on performance." Both of us looked at the HURD design and we said in effect this is not practical engineering, this is computer science masturbation. That was also, quite symbolically, the first time I met Linus Torvalds. (Raymond "Interview")

Linux: Just for Fun

Unbeknownst to the GNU people and just about everyone else, in 1991 a 21-year-old computer science student at Helsinki University in Finland named Linus Torvalds started developing his own operating system. At the university, Torvalds had been introduced to Unix, and in his spare time he had started playing with a small Unix-like system called Minix that enjoyed a fair amount of popularity at the time. Minix was a free Unix clone for personal computers written by the Dutch computer science professor Andrew Tanenbaum and was first introduced in January of 1987. It was designed to teach students about operating system design and principles, and for this reason it also came complete with source code. The rising popularity of Minix was clearly manifested in the Usenet news group comp.os.minix, where users from all over the world gathered

to discuss the system, report bugs and suggest improvements. For Tanenbaum, the increasing amount of interest in Minix on the Internet, however, was a mixed blessing. He explains:

> MINIX had a 40,000-person newsgroup, most of whom were sending me email every day. I was going crazy with the endless stream of new features people were sending me. I kept refusing them all because I wanted to keep MINIX small enough for my students to understand in one semester. My consistent refusal to add all these new features is what inspired Linus to write Linux. (Tanenbaum)

When Torvalds' feedback and suggestions for improvements of Minix went largely unanswered by Tanenbaum, he thus decided to take matters into his own hands. Postings on comp.os.minix suggests that he had started thinking about doing his own operating system sometime in the spring and early summer of 1991, but it wasn't until August that year that he made his intentions clear (Torvalds "Linux History").

> From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
> Newsgroups: comp.os.minix
> Subject: What would you like to see most in minix?
> Summary: small poll for my new operating system
> Message-ID: <1991 Aug25.205708.9541@klaava.Helsinki.FI>
> Date: 25 Aug 91 20:57:08 GMT
> Organization: University of Helsinki

> Hello everybody out there using minix -
>
> I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).
>
> I've currently ported bash (1.08) and gcc (1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)
>
> Linus (torvalds@kruuna.helsinki.fi)
>
> PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.

As opposed to the GNU HURD and Minix kernels, which were both based on a modern microkernel design, Torvalds' kernel, which later was named Linux, was based on a more traditional monolithic design. Most operating systems up until that time, including Unix, VMS, MS-DOS and others were monolithic designs where process management, memory management, i/o, and file handling, etc., were all contained in one single monolithic file. The hacker Torvalds' motivation for building his kernel on a monolithic design had

Linus Benedict Torvalds. Creator of Linux. Photo courtesy of Linux Online.

primarily to do with concern for efficiency and the processing power of the personal computers for which he targeted his system. A monolithic kernel, even though it was not state-of-the-art in system design was, in Torvalds' opinion, simply faster and more efficient than a microkernel for small personal computers. He says, "I am a pragmatic person, and at the time I felt that microkernels (a) were experimental, (b) were obviously more complex than monolithic Kernels, and (c) executed notably slower than monolithic kernels" (Torvalds "Linux Edge"). Tanenbaum, the computer science professor, was not impressed, though, when he later learned about Linux. In his view, it represented "a giant step back into the 1970s," something he explains by adding, "[it's] like taking an existing, working C program and rewriting it in BASIC. To me, writing a monolithic system in 1991 is a truly poor idea" ("Linus vs. Tanenbaum").

Linus Torvalds, however, would not be deterred by such professorial criticism, and in the middle of September 1991, shortly after the initial announcement on comp.os.minix, he released Linux version 0.01 on the Internet. Torvalds had asked people for feedback on things they wanted to see in a new operating system, and in a posting to the comp.os.minix news group on October 5, 1991, he went one step further and asked people to send him contributions that he could include in the Linux system. He opened his post with the following words:

> Do you pine for the nice days of minix-1.1, when men were men and wrote their own device drivers? Are you without a nice project and just dying to cut your teeth on an OS you can try to modify for your needs? Are you finding it frustrating when everything works on minix? No more all-nighters to get a nifty program working? Then this post might be just for you :-) (Torvalds "Linux History")

The Minix hackers must really have been ready for a change of pace because Torvalds soon had a small and dedicated following. By Christmas 1991, in the span of just four months, Linux had reached version 0.11. One could already hear the echoes of what would become the mantra of open source development in the 1990s: release early and often. Unlike Richard Stallman and the older hackers that I have presented in this chapter, Linus Torvalds, born in 1969—the same year ARPANET was first commissioned by the U.S. Department of Defense—grew up with the Internet. For him it was a natural arena in which to operate and find collaborators. While BSD had been largely a Berkeley-centered effort that

proliferated mostly in academia due to its focus on mini machines such as the PDP and VAX, Linux found its primary audience among PC hackers on the Internet. As the system became more and more functional through frequent releases and updates, it began to gain momentum and its user base soon numbered in the thousands. Unlike Unix, BSD, Minix and other systems written by hackers in academia or by professional computer scientists like Tanenbaum (who seemed to accept Linux only because "it will get all the people who want to run MINIX in BSD UNIX off my back," and who, tongue-in-cheek, chided Torvalds saying, "be thankful you are not my student. You would not get a high grade" ("Linus vs.")), Linux was an operating system written by a hacker for hackers. Only a year after Torvalds had first started toying with the idea of writing an operating system, Linux had already outgrown his expectations. In a post to the email list "Linux-Activists" in May of 1992, Linus Torvalds explains what motivated him and how it all came about. In addition to its historical interest, the post is also remarkable in the way that it offers unique glimpses into the hacker's mindset (Torvalds "Linux History"):

> To: Linux-Activists@BLOOM-PICAYUNE.MIT.EDU
> From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
> Subject: Re: Writing an OS - questions !!
> Date: 5 May 92 07:58:17 GMT
>
> [...] I didn't start off to write a kernel, I just wanted to explore the 386 task-switching primitives etc, and that's how I started off (in about April-91). [...] The first thing written was the keyboard driver: that's the reason it's still written completely in assembler (I didn't dare move to C yet - I was

still debugging at about instruction-level). After that I wrote the serial drivers, and voila, I had a simple terminal program running (well, not that simple actually). It was still the same two processes (AAA..), but now they read and wrote to the console/serial lines instead. I had to reboot to get out of it all, but it was a simple kernel. After that it was plain sailing: hairy coding still, but I had some devices, and debugging was easier. I started using C at this stage, and it certainly speeds up developement. This is also when I start to get serious about my megalomaniac ideas to make "a better minix that minix". I was hoping I'd be able to recompile gcc under linux some day... The harddisk driver was more of the same: this time the problems with bad documentation started to crop up. The PC may be the most used architecture in the world right now, but that doesn't mean the docs are any better: in fact I haven't seen /any/ book even mentioning the weird 386-387 coupling in an AT etc (Thanks Bruce). After that, a small filesystem, and voila, you have a minimal unix. Two months for basic setups, but then only slightly longer until I had a disk-driver (seriously buggy, but it happened to work on my machine) and a small filesystem. That was about when I made 0.01 available (late august-91? Something like that): it wasn't pretty, it had no floppy driver, and it couldn't do much anything. I don't think anybody ever compiled that version. But by then I was hooked, and didn't want to stop until I could chuck out minix. [...] I got bash and gcc eventually working under 0.02, and while a race-condition in the buffer-cache code prevented me from recompiling gcc with itself, I was able to tackle smaller compiles. 0.03 (October?) was able to recompile gcc under itself, and I think that's the first version that anybody else actually used. Still no floppies, but most of the basic things

worked. After 0.03 I decided that the next version was actually useable (it was, kind of, but boy is X under 0.96 more impressive), and I called the next version 0.10 (November?). It still had a rather serious bug in the buffer-cache handling code, but after patching that, it was pretty ok. 0.11 (December) had the first floppy driver, and was the point where I started doing linux development under itself. Quite as well, as I trashed my minix386 partition by mistake when trying to autodial /dev/hd2. By that time others were actually using linux, and running out of memory. Especially sad was the fact that gcc wouldn't work on a 2MB machine, and although c386 was ported, it didn't do everything gcc did, and couldn't recompile the kernel. So I had to implement disk-paging: 0.12 came out in January (?) and had paging by me as well as job control by tytso (and other patches: pmacdona had started on VC's etc). It was the first release that started to have "non-essential" features, and being partly written by others. It was also the first release that actually did many things better than minix, and by now people started to really get interested. Then it was 0.95 in March, bugfixes in April, and soon 0.96. It's certainly been fun (and I trust will continue to be so) - reactions have been mostly very positive, and you do learn a lot doing this type of thing (on the other hand, your studies suffer in other respects :)

Linus

The Little OS That Could

The Linux operating system that Torvalds invented in 1991-92 was a simple hacker's system that might very well have faded into obscurity. It had a working kernel, just a few essential utilities and tools, and it only ran on Intel X86

hardware. By 1998, however, Linux had become a well-known player in the operating systems world. It had morphed into a major multi-platform OS that had captured a good portion of the server market and that was starting to make inroads into the lucrative desktop market. How did this happen? How does one explain the Linux phenomenon of the 1990s? There is no simple answer, and the reasons are as complex and manifold as the operating system itself. Part of the answer, no doubt, is found in the resurrection of the open source concept by, among others, the BSD group and the Free Software Foundation in the 1980s, and since then manifested through the hard and dedicated work of thousands of hackers all over the world. Important parts of the answer must also be sought in the remarkable growth of the Internet itself, the economic dot.com boom in the latter part of the 90s, and the commercialization efforts within the Linux community at that time. I also believe one must seek answers in the 1990s growing dissatisfaction with the increasingly monopolistic behavior of Microsoft combined with the lure of anti-establishment hacker subculture. I will return to a more in-depth analysis of these factors later. In concluding this chapter, however, I'm going to focus on one significant reason behind Linux' success; it was a technology built on the hacker movement's successful adaptations of the time-tested Unix technology and was enhanced and expanded upon by its own users in an intricate collaborative effort.

In his autobiography, Linus Torvalds describes himself as an "accidental revolutionary" (Torvalds *Just for Fun*). I believe this is a good characterization. When he started hacking Linux in the early 90s, Torvalds had no intention of creating a major new operating system that, by the end of the decade, would compete head to head with commercial and industrial strength heavyweights like Microsoft Windows NT and Sun Solaris. All he wanted to do was to hack

and learn Unix and to create what in his mind would be a better version of Minix that he could play with on his PC. What he didn't realize, however, was that by developing the Linux kernel, he had incidentally also filled in the missing piece of the GNU puzzle. Initially, Torvalds had released Linux under a license that forbade people to make profit from it. That license, however, was soon replaced with the well-publicized GNU General Public License, according to Torvalds, because the GNU's GCC compiler he had included in Linux used the GPL (Torvalds "Linux Edge"). By licensing Linux under the GPL he not only adopted GNU's free software philosophy that encouraged people to use, copy, modify, and redistribute his system, he also tied Linux to the whole GNU operating system framework. Linux was more than just a realization of Richard Stallman's GNU OS, however. Components such as the X Windows system developed at MIT were added early on, as were numerous other components from BSD. Spawned by the successful application of the collaborative open source development model as demonstrated in the case of Linux, the mid-1990s saw a host of new projects develop that would boost Linux's usability and popularity even further. Among these we find the K Desktop Environment (KDE), the Apache web server and many, many others. These projects were started chiefly by hackers who had found Linux to be an excellent environment in which to do and promote development for both open and proprietary Unix-like systems.

In the early days of 1991-92 people who wanted to try Linux had to go on the Internet, find and download all the components that were needed, build the disk file system by hand, and compile and install the software file by file. Needless to say, this was a form of distribution that only the most ardent hacker could enjoy. In 1993, therefore, an early Linux adopter from the comp.os.minix news group, named Peter MacDonald, decided to put together a Linux distribution that

would be easy to install for new users. Characteristically he called his distribution Softlanding Linux Software (SLS). With SLS, users got an easy to use installation program that took them step-by-step through the process of installing Linux on their system. In addition to the basic Linux system that consisted of the kernel, the shared libraries, and basic utilities, users could also easily install any number of optional software packages such as the GNU development tools, Emacs and other editors, networking and mail handling software, the X Window System, and more. For convenience sake, this and other distributions that followed could also be bought on CD-ROM for a nominal cost. With fairly easy to install distributions such as SLS, one of the major thresholds for wider adoption of Linux had been surmounted. Linux was now suddenly an operating system also for non-hackers. During the 1990s the Linux distribution business grew by leaps and bounds. Carried forth by volunteer efforts such as SLS, Slackware, and Yggdrasil in the first half of the decade, to major commercial Linux distributors such as Red Hat, S.u.S.E., Caldera, and Mandrake toward the end of the decade, Linux became the most successful and widely recognized hacker effort ever. More than an operating system, today Linux is a collaborative patchwork stitched together by the philosophy of free software and the concept of open source.

Conclusions

The case study in this chapter was designed to explore the hacker movement's successful adaptation and re-engineering of technology in the period between 1970 and 2000. During this time, hackers not only took existing operating systems technology, refined it and produced their own systems and solutions such as the BSD, GNU and Linux operating systems, in the process they also invented both a

renewed philosophy of code sharing and new collaborative development models based on the concept of open source. From the earliest efforts to develop time-sharing operating systems, the desire to create environments in which programming could take place as a communal activity was an important motivator. The word time-sharing itself reflects this desire better than anything, and in short order, small hacker communities based on the sharing of code and experiences emerged around the early experimental installations at places such as MIT, Bell Labs, Berkeley, and elsewhere.

The Unix operating system, although developed within the framework of a traditional research institution was, for all intents and purposes, a hacker creation. Its simple, yet flexible and totally open structure appealed to hackers' sense of style and hands-on approach, and during the 1970s and 80s it became the OS of choice for most serious hackers. Because the AT&T telephone company in the 1970s was prevented by government order from commercializing Unix, the hackers at Berkeley were able to play with it, modify and add to it as much as they wanted. By the time the AT&T monopoly fell in 1984, and the company began to impose proprietary restrictions on it for commercial gain, the hackers had already appropriated Unix and made their own version of it: BSD. The cat was out of the bag, and it could not be put back in even though AT&T in the early 1980s went to the courts in an attempt do so.

During the 1980s, as computers in general, and the PC in particular, helped facilitate an accelerated computerization of society, the hackers' ideals of freedom and code sharing came under increasing pressure from commercialization efforts by companies and individuals who saw software as a promising new industry. In a response to these developments, Richard Stallman, once named "the last of the true hackers" (Levy 451), founded the Free Software Foundation and embarked

upon an ambitious crusade to create a free alternative operating system that hackers and others could use, share, modify, and distribute freely. GNU, as the system was called, became an important stepping stone for many hacker activities and projects in the late 1980s and early 90s, among them one by a young unknown hacker from Finland named Linus Torvalds.

As a hobby, Torvalds started out trying to create a Unix-like operating system for his personal computer. Incidentally, he ended up creating the very locomotive for most of the important hacker activities in the 1990s. A major reason for Linux's success was, as I have discussed, the fact that Torvalds was able to build his system mostly from bits and pieces that other hackers had already made. Another equally important reason was that Torvalds, quite unintentionally, was able to harness the collaborative and communicative spirit of the new Internet and thereby capture the imagination of a new generation of hackers. The popularization of the hacker movement in the 1990s is the topic for the next chapter.

4

We Changed the World
The Hacker Movement in the 1990s

When you see URLs on grocery bags, on billboards, on the sides of trucks, at the end of movie credits just after the studio logos—that was us, we did that. We put the Internet in the hands of normal people. We kick-started a new communications medium. We changed the world. –Jamie Zawinski

In 1990 few people outside of the hacker circles had heard of Free Software and the concept of source sharing. Even fewer thought there would be room for such "radical" ideas in the rapidly evolving commercial world of software engineering. By the turn of the century the ideas of Free Software and Open Source were almost on everyone's lips, and the hacker movement's leading figures and evangelists, such as Richard Stallman, Linus Torvalds, Eric Raymond and others, had acquired fame and status far outside their home turf. The hackers were no longer the anti-social misfits that the popular media loved to talk about every time there was a computer virus on the loose or a data break-in. They were the new heroes—the creative entrepreneurs of the dotcom era and their ideas, ethics and methods changed not only the landscape of software engineering but also ushered in new business models and products that challenged the

establishment. In the 1990s the hackers were the ones who realized they had a shot at changing the world, and they went for it.

In this chapter I will discuss some of the significant developments that furthered the rise of the hacker movement in the 1990s. The chapter starts with a presentation of Red Hat Inc, one of the companies that made a successful business out of Linux and thereby showed the world that one could make money on "free software." I then move on to look at the relationship between the Free Software and Open Source movements and how these came to stimulate the adoption of hacker development methodologies in the software industry. This serves as a backdrop for the second part of the chapter where I focus specifically on two projects, Apache and Mozilla, that helped cement the acceptance of Free and Open Source software as viable and enduring alternatives to proprietary models and practices. The chapter concludes with an analysis of three of the most important online hacker communities of the late 1990s and discusses how these helped broaden and strengthen the hacker movement in various ways.

In Search of "The Next Big Thing"

The widespread adoption of the Internet and the communication technologies associated with it contributed in large part to a further globalization of the hacker movement in the 1990s. In the 1960s and 70s, hackers typically worked in small groups or as individuals in relative isolation from one another. In the 1980s, communication technologies such as Email, IRC, Usenet and later the Web, came to facilitate the formation of new and geographically diverse groups of hackers. Linux, a main case in the previous chapter, is an example of a technology born in this new environment. Hackers such as Linus Torvalds, however, were not the only ones to discover the new possibilities the Internet had to offer. In the years

between 1994 and 1998, a vast number of start-ups and established companies alike flocked to the Internet hoping to capitalize on what they perceived as the "next big thing." While this move was made possible by deregulation and subsequent decommission of the old ARPANET in 1990, for most it was the World Wide Web that held the promise of golden business opportunities. One company that would use the power of the Internet to create a business out of Free Software was Red Hat Software. I will shortly return with a discussion of this company and how it fits into the larger picture of the hacker movement in the 90s. First, however, I want to clarify a few points about the Internet and how it relates to the hacker movement specifically.

Collaboration. The Internet represented a worldwide communication infrastructure that let people with converging interests connect across geographical barriers and time zones, across age differences and social status. The Internet constituted a worldwide arena for collaboration.

Communication. The Internet represented a highly versatile and up-to-date source of information about new developments and interesting projects. It provided the perfect avenue for spreading the "gospel of free software." The Free Software Foundation was slow to catch on to the significance of the Internet in this regard. Linus Torvalds, on the other hand, may not have completely understood the ramifications of using the Net, but according to Eric Raymond, for him, using it was second nature.

Distribution. The Internet represented an effective medium for distribution of software. It made it possible to share data and software at no, or minimal, cost and at the same time reach a worldwide audience.

As the Web matured in the 1990s, it quickly came to be seen as a new frontier for business opportunities. The rise of the dotcom economy in the second half of the decade became an important catalyst for the growth of the issues I am discussing. Everyone was falling over themselves to have a presence on the World Wide Web. Everyone tried to leverage the promise of the web to benefit their own cause or goal, so why shouldn't the proponents of the free software movement do the same?

Red Hat: "Rising on a wave of inevitability"

"Rising on a wave of inevitability"—these were the words that Red Hat's own web site used in 2002 to describe the company's rise to fame in the late 1990s (Red Hat Linux). While the increasing popularity of, and focus on, Free Software during this period was by no means "inevitable," as the corporate lore seems to believe, Red Hat company managed to tap into what was fast becoming one of the hottest commodities in the software business and was thus well positioned to ride the Open Source wave that soon followed.

Robert Young and Marc Ewing founded Red Hat Software in January of 1995. Young had started out with a small software distribution business in the early 90s and had followed the early developments of Linux with great interest. Contrary to his expectations, however, the Linux phenomenon proved to be more than just a fluke, and with its rapidly growing popularity he became more and more intrigued by what business opportunities they could garner from it. Young explains:

> We found that instead of this bizarre Linux OS effort collapsing, it continued to improve. The number of users continued to grow and the

> applications they were putting it to were growing in sophistication. So we began to study the OS development more carefully. We spoke to the key developers and the largest users. The more we studied, the more of a solid, albeit unusual, economic model we saw. (Young)

As I explained in the previous chapter, the Free Software distribution business was well underway by the mid 1990s with more than a dozen different entities offering Linux or BSD solutions for sale on CDs or free downloads via FTP. However, while most of these developers focused their efforts on the technical aspects of compiling and building the software distributions, Young and his partner, Marc Ewing, realized that as the number of people using Free Software continued to grow there would also be a market for consulting services and logistics- and end-user support. He explains:

> The "unique value proposition" of our business plan was, and continues to be, to cater to our customers' need to gain control over the operating system they were using by delivering the technical benefits of freely-redistributable software (source code and a free license) to technically-oriented OS consumers. (Young)

Young compares Red Hat's business model to that of the auto industry. No car manufacturer makes all the parts that make up their various models; instead, he says, they buy their parts from a number of different sub contractors. The automaker's job is to put all these different parts together and make a car out of them. In a similar fashion, Red Hat uses a number of different software components in their retail distributions. At the core there is the Linux kernel with

libraries and the GNU development tools. The user interface may be based on KDE in one release and GNOME in another. Finally there is a suite of application programs for all sorts of purposes including image manipulation, database development and office tools. None of these components come from Red Hat's own engineers; instead, they are collected from a variety of sources within the Free Software world. Red Hat's job is to put them all together in a coherent way necessary to create a complete and functioning operating system that will satisfy end users. In later years Red Hat has also started to contribute with software components of their own, most notably the Red Hat Packet Manager (RPM), a program designed to make it easy to install/uninstall and upgrade various components in Red Hat Linux.

Until Red Hat entered the Free Software distribution business in the mid 1990s, the Linux operating system could not be considered readily available to the general public. It was mostly an online phenomenon and something for the technical elite who knew where to find it, and who could tweak and work on their own systems. When the Red Hat Linux distributions started to appear on the shelves of software retailers, however, something significant had happened. For the first time, a major Free Software product was being marketed and distributed alongside commercial offerings. A prospective user could walk into any number of retail stores and walk out with a copy of Linux neatly wrapped and packaged and complete with printed manuals and telephone and online support.

Unlike other Linux distributors at the time, which were either private individuals or small non-profit organizations, Red Hat was a commercial company that had just made a business out of selling Free Software. When the company first started in 1995, this sounded to most industry observers like an

oxymoron, but over the next few years, as more and more companies followed suit, it became apparent to pundits and media analysts that Free Software was indeed the Next Big Thing.

When Red Hat Software Inc. went public in 1999, the company raised $84 million in its first day of trading and came in among the top performing IPOs on Wall Street that year (Glasner "The Biggest IPO"). Another company, VA Linux, set a new record for the largest first-day gain in IPO history, with a 733 percent run-up when it went public in December the same year. Analysts at the time explained these astounding figures by "the general hoopla among investors about anything remotely tied to the Linux operating system" (Glasner "VA Linux"). This was at the very eve of the dotcom era, but nevertheless, in eight short years, the Free Software movement propelled by Linux had gone from an obscure hacker phenomenon to a multi-million dollar enterprise. The hackers had become the new stars on Wall Street, and Linus Torvalds himself was the brightest of them all. To fully understand how this astounding achievement was possible, it is necessary to step back a few years and look at some other important developments that took place on the Free Software scene in the interim.

"The Cathedral and the Bazaar"

As word of the new Linux OS spread rapidly throughout the hacker movement in the early 1990s, it attracted users, testers and contributors in droves. One of these new Linux converts was Eric Steven Raymond, also known online as ESR. Raymond had first become involved with the hacker movement in the late 1970s, and in the mid 1980s he had been one of the earliest contributors to the Free Software Foundation's GNU project. Another of his early and important contributions was as editor and maintainer of what was then called the *Jargon*

File, an online collection of hacker lore, slang and jargon (Raymond, "New Hacker's"). When Raymond first discovered Linux in early 1993, he realized that something new and big was afoot within the hacker movement. Inspired by Linus Torvalds' new and creative development model, he thus began a series of theoretical writings designed, as he says, to "illuminate the social patterns and development methods that that culture was already using effectively, in a way that made it possible for other people to understand and emulate them" (Raymond "Interview"). The most influential of these writings was a paper entitled "The Cathedral and the Bazaar" (1997).

"The Cathedral and the Bazaar" attempts to describe and understand the Linux approach to software engineering. In traditional software engineering, which Raymond likens to the classic cathedral building of the Middle Ages, a program, he says, is typically crafted by a group of highly skilled craftsmen presided over by a master builder who provides the vision and the grand plans for the project. He explains:

> Linux overturned much of what I thought I knew. I had been preaching the Unix gospel of small tools, rapid prototyping and evolutionary programming for years. But I also believed there was a certain critical complexity above which a more centralized, a priori approach was required. I believed that the most important software (operating systems and really large tools like the Emacs programming editor) needed to be built like cathedrals, carefully crafted by individual wizards or small bands of mages working in splendid isolation, with no beta to be released before its time. (Raymond "Cathedral")

When he looked at the way in which Linus Torvalds had organized the Linux project, on the other hand, none of these "truisms" were anywhere to be found. On the contrary, to Raymond the Linux engineering model looked most of all like a "great babbling bazaar" in which basically anyone could bring their goods to market. If the project leaders deemed a particular contribution significant and valuable enough, it would then make its way into the code and become part of the bigger mosaic.

> Linus Torvalds' style of development—release early and often, delegate everything you can, be open to the point of promiscuity—came as a surprise. No quiet, reverent cathedral-building here—rather, the Linux community seemed to resemble a great babbling bazaar of differing agendas and approaches (aptly symbolized by the Linux archive sites, who'd take submissions from anyone) out of which a coherent and stable system could seemingly emerge only by a succession of miracles. The fact that this bazaar style seemed to work, and work well, came as a distinct shock. (Raymond "Cathedral")

To test his theory and to see for himself how the bazaar model would work in a practical setting, Raymond set out to create a new Free Software project based on Torvalds' Linux model. Raymond had long held the belief that "every good work of software starts by scratching a developer's personal itch" (Raymond "Cathedral"), so consequently he chose an email POP3 client that he had wanted for some time as the test bed for his project. The POP3 client, later named Fetchmail, would only be a bi-product of the much more important paper that eventually came out of the project. In this paper, Raymond summarizes some of

the lessons from the bazaar experiment with the following key observations: (Raymond "Cathedral")

1) Every good work of software starts by scratching a developer's personal itch.
2) To solve an interesting problem, start by finding a problem that is interesting to you.
3) Good programmers know what to write. Great ones know what to rewrite (and reuse).
4) Release early. Release often. And listen to your customers.
5) If you treat your beta-testers as if they're your most valuable resource, they will respond by becoming your most valuable resource.
6) When you lose interest in a program, your last duty to it is to hand it off to a competent successor.

"The Cathedral and the Bazaar," (subtitled "Why does the Linux development model work?") was presented publicly at the 4th International Linux Kongress in Würzburg, Germany in 1997. It was warmly received by the conference audience and soon made headlines on Linux and Free Software web sites all across the Internet. Raymond had actually succeeded beyond his expectations in his theoretical undertaking. He had managed to dissect the Linux phenomenon and describe its inner workings and benefits in ways that made the hacker community's implicit knowledge of its own practices explicit. Furthermore, he had managed to do this in a way that people outside of the immediate hacker circles could understand. This last point would soon prove to be one of the most crucial upshots of the paper.

In Mountain View, California, there was a company with a problem. They had once presided over the most successful web browser in the world, but in recent years their hegemony had come under severe attack from their most ardent competitor. They were looking for ways to remedy the unfortunate predicament in which they found themselves, and they believed that Eric Raymond's bazaar model presented the salvation. That company was Netscape Communications Corporation (Leonard "Let my"). In January of 1998 in an unprecedented move that surprised most industry observers, Netscape announced that it would release the source code to its popular but ailing web browser to the Free Software community ("Netscape Announces"). The so-called browser wars that had raged with increasing intensity since Microsoft had first released its Internet Explorer web browser in 1995 had left Netscape's market share trailing far behind at this point. It was believed that by opening up the source and letting independent programmers in, Netscape could once again gain the upper hand in the struggle for market dominance. According to their 1998 news release, Netscape hoped that this would enable them to "harness the creative power of thousands of programmers on the Internet by incorporating their best enhancements into future versions of Netscape's software" ("Netscape Announces"). The decision to "free the source" eventually led to the creation of the Mozilla project. After Linux, this was perhaps the biggest and most influential Free Software project of the late 1990s and early 2000s.

The Open Source Initiative

In spring of 1998 a new buzzword was beginning to be heard on the Internet grapevine. The word was *Open Source*. An article in the online *Wired News* from May 1998 opens with the following passage: "If you've sent email or browsed a

Web site, you've almost certainly come into indirect contact with "open source" software without even knowing it" (Glave). The author makes only a fleeting reference to the term in this article, but before the end of the year the new phrase Open Source had all but replaced Free Software in the public discourse. To the uninformed, the distinction between the two may seem semantic at best. In reality, however, while describing more or less the same technical concepts and development processes, the two phrases represent two almost entirely different philosophical views on software development.

When Richard Stallman first coined the term Free Software back in the mid-1980s, he wanted it to say something about personal freedoms to create, use, change and redistribute software. To him, it was a moral imperative. "For the Free Software movement, non-free software is a social problem and free software is the solution," says Stallman ("Why 'Free'"). In the English language, the word *free* also commonly refers to something you can get *for free*, or *free of charge*. This double meaning of the word turned out to be somewhat unfortunate, and Stallman later claimed that it was never his intention that Free Software should also be *gratis*. He explains: " 'Free software' is a matter of liberty, not price. To understand the concept, you should think of 'free' as in 'free speech,' not as in 'free beer'" (Stallman "Free Software Definition").

In the late 1990s, some people, including Eric Raymond, started to feel that the adoption of Linux and other Free Software programs, especially within the large and important corporate sector, was hampered by the very label Free Software that many, especially outside the hacker movement, frequently mistook for free as in *gratis*. Raymond elaborates:

> I became aware that our largest problem as a culture was not technology but image. We had created a perception of our work and our goals and our social organization that was hurting us badly and hindering the adoption of our technology and getting in the way of our goals. In particular I realized that the term Free Software was a problem. It was not a problem because of what it means inside our community, but because of what it promotes outside. (Raymond "Interview")

When Raymond first learned of Netscape's decision to open the source to their browser software, he realized immediately that a golden opportunity had just opened up for the hacker movement. In an interview with online magazine *Salon.com* shortly after Netscape's announcement he says:

> We knew we had a better way to do things in our software designs and operating systems and the way that we shared work with each other. But we couldn't get anybody to listen. Netscape doing this creates a window of opportunity for us to get our message into corporate boardrooms. The flip side of that is that if Netscape tanks, no one is going to listen to us for another decade. (Leonard "Let my")

There was suddenly a lot at stake, and Raymond wasn't about to let an historical chance go by the wayside. In his own words, "it was time to stop being an accidental revolutionary (a reference to Linus Torvalds' autobiography) and start being a purposeful one" (Raymond "Interview"). On February 3rd 1998, he met with Todd Anderson and Chris Peterson of the Foresight Institute, John "maddog" Hall and Larry Augustin of Linux International, and Sam Ockman of

Silicon Valley Linux User's Group in Palo Alto, California (OSI "History"). At this meeting the group brainstormed various strategies for how best to sell the bazaar-style software engineering models to the corporate world, which, they were convinced, was finally ready to listen now that Netscape had given the all-important imprimatur.

> We realized it was time to dump the confrontational attitude that has been associated with "free software" in the past and sell the idea strictly on the same pragmatic, business-case grounds that motivated Netscape. We brainstormed about tactics and a new label. "Open source," contributed by Chris Peterson, was the best thing we came up with. (OSI "History")

Not long after the meeting, Eric Raymond and Bruce Perens co-founded the Open Source Initiative (OSI), a non-profit organization dedicated to promote the label Open Source and a new set of licensing guidelines dubbed the Open Source Definition (OSD). A more thorough analysis of the OSD will follow later, but for the purposes of the historical context here, however, I shall make just a few remarks on what it is and how it differs from regular licenses such as the GNU GPL or the BSD license. Bruce Perens drafted the original document using the *Debian Free Software Guidelines* (of which he was also the primary author) as a template. Thus, the OSD is not a software license per se; rather, it is a set of guidelines and criteria that an Open Source software license must meet.

The founders of the OSI wanted Open Source to have the broadest possible appeal, especially within the important commercial world; and thus, as opposed to the GNU GPL, there is no reference to the troublesome word *free* anywhere in the OSD document. More importantly, however, while software licensed under

the GNU GPL is largely incompatible with proprietary solutions, the OSD was designed specifically to let companies incorporate open source solutions within their proprietary software and vice versa. In this respect, the OSD takes its cue from the BSD license in which no restrictions are placed upon proprietary components. An important stipulation in the OSD is, of course, that programs licensed under an OSD-type license must include source code, or, the source code must be readily available upon request. Also, any work under the OSD guidelines must allow for modification and derivative works as well as redistribution of those works. By the year 2002, more than 40 different types of software licenses were said to conform to the guidelines set out by the OSD. These included licenses from several major corporate actors in the software business such as Apple Computer, Corel, IBM, AOL/Time Warner, Troll Tech, Sun Microsystems, Nokia, and Sybase (OSI). The vast majority of Free Software and Open Source projects, however, continued to use variations of the GNU GPL or the BSD license.

A Conflict of Interests: Ideology vs. Pragmatism

Not long after the formation of the Open Source Initiative, a conflict that had been latent within the hacker movement for the better part of the 90s suddenly came to a head. On one side of the divergence were the proponents of Richard Stallman and his ideology of Free Software as expressed through the GNU GPL and other writings published by the Free Software Foundation. On the other side were the proponents of the less restrictive, more pragmatic BSD-style licensing scheme, which the new Open Source Initiative sought to embrace. The conflict was complex and had several different layers to it as Eric Raymond explains from his point of view:

> There is an ideological level, there is a practical level, and there is a personal level. On the ideological level, what you are seeing here is a conflict between two ideologies, two value systems that lead to similar behaviors but justify and advocate those behaviors in fundamentally different ways. The difference in ideology is fairly clear. Richard's mode of changing the world is very consciously and explicitly a moral crusade. If you don't buy his abstract moral arguments, you don't sign up. The way that Linus Torvalds and I, and others, have pushed is a pragmatist's way. In effect what we're saying is; "Moral argument? Who needs a moral argument? We like engineering results!" So there is a level at which it is a conflict about what kind of ideology we are going to have.
>
> There is a practical level on which it is a conflict about which kinds of tactics work for spreading our message. The argument on that level is; Does it work better to moralize or talk about results?
>
> There is also a personal level in that, well, people have egos and they have investments in their ideologies, and if part of what's going on is that you think somebody hijacked your revolution, you kind of resent that. (Raymond "Interview")

Stallman's radical political agenda, coupled with his well-known unwillingness to compromise on the premise of Free Software, meant that there was, in effect, no room for him in the new Open Source movement. In their attempts to sell their ideas to the business world, therefore, the Open Source

Bruce Perens. Self-taught Unix hacker, author of the Open Source Definition, and co-founder of the Open Source Initiative. Photo courtesy of Gary Wagner. © 2001 garywagnerphotos.com

advocates purposefully distanced themselves from Stallman and Free Software. The last thing they wanted was to alienate potential partners in business and industry by what they saw as Richard Stallman's "infamous" confrontational attitude. Stallman on his side did little to ease the tension. To the contrary, in response he boldly turned up the rhetoric a few more notches by accusing the Open Source proponents of "neglecting political issues, and jumping on the bandwagon of the day," which, he believed to be, "a common form of shallowness in our society" (Leonard "Stallman saga"). To further drive his

argument home, Stallman later published a paper in which he outlined what he believed to be the fundamental differences between Free Software and Open Source. One of the opening passages of his paper reads as follows:

> The fundamental difference between the two movements is in their values, their ways of looking at the world. For the Open Source movement, the issue of whether software should be open source is a practical question, not an ethical one. As one person put it, ``Open source is a development methodology; free software is a social movement.'' For the Open Source movement, non-free software is a suboptimal solution. For the Free Software movement, non-free software is a social problem and free software is the solution. (Stallman "Why 'Free'")

In another position statement entitled "Freedom or Power?" Stallman elaborates further upon the notion of Freedom. He says:

> In the Free Software Movement, we stand for freedom for the users of software. We formulated our views by looking at what freedoms are necessary for a good way of life, and permit useful programs to foster a community of goodwill, cooperation, and collaboration. […]
>
> We stand for freedom for programmers as well as for other users. […] However, one so-called freedom that we do not advocate is the "freedom to choose any license you want for software you write." We reject this because it is really a form of power, not a freedom. […] If we confuse

> power with freedom, we will fail to uphold real freedom. (Stallman and Kuhn "Freedom")

Although the conflict between the ideologists and the pragmatists within the hacker movement eventually quieted down, it had for all practical purposes created two separate wings, a left and a right as it were, within the broader movement. Many, even within the Open Source Initiative, thought this schism was an unfortunate turn of events. Bruce Perens, for instance, had originally envisioned the Open Source Initiative as a gentle introduction to the deeper philosophy of Free Software, rather than a separate movement; and for this reason he soon came at odds with Eric's Raymond's separatist hard-line. In an interview with *Linux Magazine* in September, 2001, Perens says:

> When we founded Open Source, my understanding was that Open Source would be a gentle introduction to Free Software and not a separate movement. I would never have participated in Open Source for the purpose of creating a schism. Especially now, it is important that we stand together. That's more important than it used to be. I harbor some disappointment that Open Source became something that sort of deprecated Richard Stallman's philosophy rather than leading people into Free Software. (McMillan "Our Man")

The two wings eventually resumed collaboration on practical projects of common interest, but their individual agendas, strategies, and goals would remain different. Unity was not the only casualty from the fallout between Open Source and Free Software. On February 18th, 1999, Bruce Perens resigned from the

Open Source Initiative. In an email distributed on the Net he said: “It's Time to Talk about Free Software Again” (Perens “It’s time”).

The Halloween Documents: The Empire Strikes Back?

By the end of the 1990s, the Linux operating system had grown into a considerably popular server platform and had also started to make inroads into the lucrative desktop market. For many new users Linux represented a form of liberation from the firm monopolistic grasp that Microsoft had established in the personal computing market. Some observers even came to believe that the astonishing Linux adoption rate was more than anything indicative of a giant backlash against the omnipresent Microsoft. Ken Thompson, creator of Unix, for instance, had this to say about Linux in 1999:

> I view Linux as something that's not Microsoft—a backlash against Microsoft, no more and no less. I don't think it will be very successful in the long run. I've looked at the source and there are pieces that are good and pieces that are not. A whole bunch of random people have contributed to this source, and the quality varies drastically. (Cooke et al “Unix and Beyond”)

With Netscape opening the source code to their browser and the new Open Source movement rapidly gaining favorability and coverage in the press, the silence and apparent inactivity on the part of Microsoft grew ever more ominous. Then, shortly before Halloween of 1998 a confidential Microsoft memorandum on Linux and Open Source was “leaked” to Eric Raymond. He immediately realized that this was pure gold for the Open Source movement and, being an

eminent writer and strategist with a flair for the dramatic, he spent the Halloween weekend annotating the Microsoft document with his own comments and analysis, whereupon he promptly released it to the national press as the "Microsoft Halloween Document." The press ran with the story, and a few days later Raymond received a second Microsoft memorandum from an undisclosed source, this one focusing specifically on Linux (Raymond "Halloween Documents"). The gist of the original Halloween document can be summarized in the following key points.

1) OSS poses a direct, short-term revenue and platform threat to Microsoft, particularly in server space. Additionally, the intrinsic parallelism and free idea exchange in OSS has benefits that are not replicable with our current licensing model and therefore present a long term developer mindshare threat.

2) To understand how to compete against OSS, we must target a process rather than a company.

3) OSS is long-term credible ... FUD tactics cannot be used to combat it.

4) Linux can win as long as services / protocols are commodities.

5) OSS projects have been able to gain a foothold in many server applications because of the wide utility of highly commoditized, simple protocols. By extending these protocols and developing new protocols, we can deny OSS projects entry into the market. (Raymond Halloween I")

As opposed to what the official Microsoft position was at the time, the document clearly shows that the company viewed Open Source software (OSS) as a threat to their short and long term market dominance. They also seemed to understand that they were not up against a traditional company or business this time, but rather, what they somewhat erroneously refer to as a process. The opponent was, of course, the entire hacker movement. The thing that perhaps scared people the most about the first Halloween document, however, was Microsoft's proposed strategies for combating OSS in the market place. Spreading FUD (Fear, Uncertainty and Doubt) about competing products and the "de-commoditization of protocols" through embrace and extend tactics were practices that many had accused Microsoft of in the past. The company had never officially admitted to such practices, however, but the Halloween document clearly spoke a different, more sinister truth.

In the spirit of the Halloween holiday, the Microsoft documents served Raymond's agenda perfectly. "The new David that the Goliath of Redmond has in its sights is the free Linux operating system and the 'open source' software development community that built it," writes Scott Rosenberg in *Salon.com* on November 4th. "War hasn't been formally declared yet but the battle plans are now a matter of public record" (Rosenberg "Let's Get").

The so-called Halloween documents had first been circulated internally at Microsoft in August of 1998. After they became public knowledge, the company admitted to their authenticity, but brushed aside any speculations as to their purpose, claiming that they were simply whitepapers meant to "stimulate internal discussion on the open source model and the operating system industry." In no way did they represent "an official Microsoft position or road

map," said Microsoft group marketing manager, Ed Muth, in an official rebuttal posted on Microsoft's web site (Microsoft "Linux"). "Company executives dismissed open-source as hype," he said, and added: "Complex future projects [will] require big teams and big capital. These are things that Robin Hood and his merry band in Sherwood Forest aren't well attuned to do" (Raymond "Halloween IV").

Regardless of what the Halloween documents may or may not have said about Microsoft's intentions, or whether Mr. Muth realized that his patronizing remarks inadvertently cast Microsoft's executives in the infamous role as the Sheriff of Nottingham, Eric Raymond, in a brilliant strategic maneuver, had managed to create a story that in the public mind situated the Open Source movement as the antithesis to the Redmond software giant.

Apache: A Patchy Server for the World Wide Web

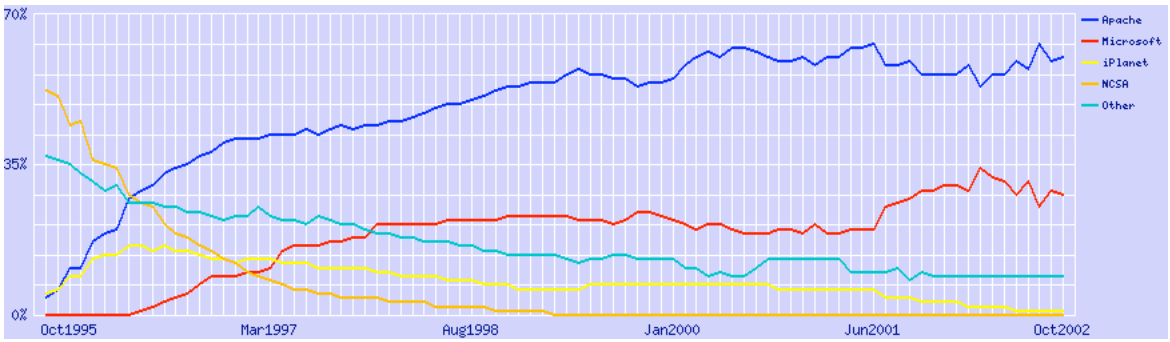
While the GNU/Linux OS, and all the various projects associated with it, was undoubtedly the locomotive of the hacker movement in the 1990s, there were many other independent projects that also sought to leverage the promise of Open Source. In what follows I will examine two projects that for various reasons and in various forms represent important milestones in the recent history of Open Source software development. These are the Apache web server and the Mozilla web browser. The analysis of these two projects is designed to uncover more details concerning the organization and inner workings of successful Open Source projects, as well as show some of the diversity that characterized Open Source development in the late 1990s and early 2000s.

In the early to mid 1990s, the most popular web server software on the Internet was the NCSA HTTPd (HTTP daemon), a freeware program developed

by Rob McCool, an undergraduate student, system administrator, and programmer at the National Center for Supercomputing Applications (NCSA). According to McCool, the HTTPd program was based on the original CERN Hypertext Transport Protocol which, "at that time was around version 0.9 or 1.0, and was very simple. I found that it wasn't the protocol itself that was interesting, but rather all the things that could be built around it" (McCool et al "Apache Story"). Thus, he began a series of modifications and extensions to the WWW server software that brought along a number of important innovations such as the URL (Uniform Resource Locator), server parsed HTML, and CGI (Common Gateway Interface). When McCool later joined Netscape in 1994, development of the HTTPd program stalled, and many web masters and server administrators began to implement their own bug fixes, patches, and extensions.

In February of 1995, some of these web masters met over email to discuss ways in which they could coordinate their independent efforts. Roy T. Fielding, one of the original members of this group and co-founder of the Apache project explains:

> A small group of these webmasters gathered together via private e-mail for the purpose of coordinating their changes (in the form of "patches"). Brian Behlendorf volunteered the use of his server as a shared information space, hosting a public mailing list for communication and providing logins for the developers, including Robert S. Thau, Rob Hartill, Randy Terbush, David Robinson, Cliff Skolnick, Andrew Wilson, and myself. (McCool et al "Apache Story")



Market Share for Top Web Servers Across All Domains August 1995 - October 2002.

Source: Netcraft Web Server Survey.

Developer	*Servers Oct 2002*	*Market share*
Apache	21,258,824	60.54%
Microsoft	10,144,453	28.89%
Zeus	711,998	2.03%
iPlanet	478,413	1.36%

The group eventually put together and released "a patchy server" based on McCool's HTTPd code, and thus, the "Apache" project was officially under way. Less than a year after its first release, Apache had become the most popular web server on the Internet.

Data from the Netcraft Web Server Survey (Netcraft) shows that by October of 2002 the Open Source Apache server had amassed a whopping 60% share of the total market for web servers. The commercial offering from Microsoft came in a distant second with just about 29% of the market share. These astounding figures, along with its long-term survival in the marketplace, have made the Apache project a poster child of the hacker movement. To understand how such an accomplishment was possible, we need to appreciate not only the characteristics of the market place, but also the Apache project's organization and development practices.

To start with the market place, it is clear that Apache got off to a flying start by essentially filling the void left behind by the orphaned NCSA HTTPd. In those early years, Netscape's Enterprise Web servers enjoyed a considerable share of the market, but as Netscape's influence sharply declined toward the end of the 90s, their servers all but disappeared from the marketplace. Microsoft, on the other hand, with their lineup of web, Intra- and Internet server solutions has, according to the Netcraft Survey, seen a steady, albeit slow growth in their market share. When looking at these numbers, it is helpful to keep in mind that the Internet in the 1990s was, to a large extent, built on Unix servers. Apache, which targeted these directly, was therefore at a considerable advantage compared to Microsoft who mainly targeted their own server platforms. According to a 1998 survey by market researcher Dataquest, Unix servers held 42.7 percent of the server market that year. By contrast, Microsoft NT's share of the market was only 16.2 percent (Patrizio).

While the Linux kernel project, as I have explained earlier, was organized in a vertical, hierarchical fashion with Linus Torvalds as its benevolent dictator, the Apache model was much more horizontal and decentralized. The core developers were all volunteers from several different countries, including the United States, the UK, Canada, Germany and Italy; and in order for the development efforts to succeed, the organizational decision making process had to be flexible enough to accommodate everyone's time zone and personal schedule. In a 1999 paper published in *Communications of the ACM*, Roy Fielding explains how the early decision making process worked:

> There was no Apache CEO, president or manager to turn to for making decisions. Instead, we needed to determine group consensus, without

> using synchronous communication, and in a way that would interfere as little as possible with the project progress. What we devised was a system of voting via email that was based on minimal quorum consensus. Each independent developer could vote on any issue facing the project by sending mail to the mailing list with a "+1" (yes) or "-1" (no) vote. For code changes, three positive votes and no negative votes (vetoes) were required before the change would be allowed in the final source. For other decisions, a minimum of three "+1" and an overall positive majority was required. Anyone on the mailing list could vote, thus expressing their opinion on what the group should do, but only those votes cast by the Apache Group members were considered binding. (Fielding)

From 1996 onward the group used CVS (Concurrent Versions System) for managing the shared code base. CVS is a version control system that allows developers in different locations to work on a shared code base. In a CVS system the code is organized in a central repository that keeps track of any changes (history) made to it. Independent developers can "check out" modules or files from the central CVS repository as well as "commit" changes to it if they are so authorized (Cederqvist). CVS allowed the Apache group to streamline their development efforts by enhancing the collaborative environment and giving each individual developer better access to the shared code. The project was organized as a "meritocracy," which meant that the more good code a person contributed, the more access privileges and responsibility they would receive (Fielding).

While Apache by any account was already highly successful, the project received a further boost when in 1998 IBM announced that the company would formally adopt Apache as their web server platform and, furthermore, come on

board as a contributor. In the beginning, members of the group were skeptical of IBM's motives and what it could mean for the project as a whole. Apache co-founder Brian Behlendorf explains:

> Most developers had some sort of financial motivation to be involved, yet overall the main driver at that point was that working on and being involved with Apache was "fun." Could it still be "fun" with an 800-pound gorilla involved? Could we reconcile the fact that up to this point, it had been a group of individuals working together, yet this was a company asking to join us? Would this mean we'd become essentially unpaid developers for someone else? (McCool et al)

After many rounds of discussions within the group, it was decided to allow IBM in. The decision would prove advantageous for the Apache team on many different levels, and according to Behlendorf, "IBM set the gold standard for how to work as a peer within an Open Source development project" (McCool et al). It is ironic, perhaps, that IBM of all companies would align themselves in this way with the hacker movement. Had it been only five or ten years earlier, such a marriage would have been unthinkable.

Mozilla: A Revolutionary Project for Revolutionary Times

> We sat in the conference room and hooked up the big TV to one of the Indys, so that we could sit around in the dark and watch the FTP download logs scroll by. jg [sic] hacked up an impromptu script that played the sound of a cannon shot each time a download successfully completed. We sat in the dark and cheered, listening to the explosions.

> Four hours later, the Wall Street Journal was delivered, and it already contained an article describing what we had just done. ``Clients aren't where the money is anyway,'' ran the quote from Marc.
>
> I'd go home now if I thought I could drive there without dying, so instead I'm going to curl up under my desk again and sleep here. Maybe we're not doomed; people on the net are talking about Mozilla with all caps and lots of exclamation points. They're actually excited about it....(Zawinski "Netscape Dorm")

The above except is from the diary of Jamie Zawinski, Netscape employee #20. The entry is dated October 12th 1994; and it was the day the company released the 0.9 version of its first browser dubbed Mozilla. Fast-forward eight years to June 5th 2002, another Mozilla hits the World Wide Web ("Mozilla 1.0"). What had transpired in the interim was nothing less than the rise and fall of Netscape Communications Corporation, perhaps the most highly profiled dotcom of the decade.

By the time Netscape announced (January of 1998) that they would release the source code to their Communicator suite, much had changed since the euphoric days of 1994 when, akin to a modern-day King Midas, everything the company touched seemed to turn gold. Jamie Zawinski explains:

> In January 1998, Netscape hit one of its blackest periods -- the first round of layoffs. It was quite a wake-up call. Netscape, darling of the computer industry, the fastest-growing company in the world, was not invincible.

> More concretely, this was when we realized that we had finally lost the so called ``browser war.'' Microsoft had succeeded in destroying that market. It was no longer possible for anyone to sell web browsers for money. Our first product, our flagship product, was heading quickly toward irrelevance. (Zawinski "Nomo zilla")

As revealed in the U.S. vs. Microsoft anti-trust proceedings in 1998 ("Testimony"), Netscape had certainly met with some questionable business tactics from latecomer Microsoft; but contrary to the official corporate spiel, their problems were as much their own doing as they were Microsoft's. Now that we have a historical perspective on the events, we can see that after its initial successes in the mid 1990s, Netscape, as a company, essentially stopped innovating. The quality of their software became a liability, and spokespersons for the company admitted in 1997 that "[t]he company's priorities used to be (in descending order) features, schedule, and quality" (Lash). When faced with an aggressive competitor like Microsoft, who was not only able to leverage the full benefits of a close integration between their Internet Explorer browser and the Windows operating system, but also was able to produce a higher quality product, it was only a matter of time before Netscape was out of the so-called "browser wars" altogether.

It is in light of these events that we must understand Netscape's decision to go Open Source. On March 31st 1998, the day that the source code to the Communicator 5.0 suite was posted on the Internet, Marc Andreessen, Netscape executive vice president of products said in a news release: "Releasing the Communicator source code rallies the developer community while leveraging the incredible talent of the net. Everyone wins" ("Netscape Accelerates").

To facilitate and coordinate the Open Source effort, Netscape set up an organization named Mozilla.org. Although several people were involved in the establishment of the organization and the project it incorporated, Frank Hecker, one of the architects behind Netscape's move to go Open Source, credits two people in particular for getting the Mozilla project off the ground. He says:

> A number of people at Netscape were instrumental in setting the Mozilla project on the right path when it initially started back in 1998. I want to mention especially Jamie Zawinski and Mitchell Baker.
>
> Jamie Zawinski did a great job of establishing an independent identity for the Mozilla project at its outset; he lobbied for creation of the mozilla.org web site independent of Netscape's site, lobbied for creation of mozilla.org as a separate group dedicated to making the Mozilla project successful, and commissioned and directed the creation of the Mozilla logo and "brand image." Most important, Jamie articulated a clear vision for the Mozilla project and for mozilla.org.
>
> Mitchell Baker and others were originally responsible for Netscape's efforts to create an open source license for the Mozilla code; this eventually resulted in the creation of the Netscape Public License and then the Mozilla Public License. (Suárez-Potts)

Mozilla pioneers: Mozilla evangelist Jamie Zawinski (left) and "Chief Lizard Wrangler" Mitchell Baker.

The Mozilla mission statement, originally authored by Jamie Zawinski, gives us a good idea of both the original vision and goals for the project, as well as pointers to its organizational structure.

1) We will provide technical and architectural direction for the project.

2) We will collect changes, help authors synchronize their work, and periodically make new source releases which incorporate the best work of the net as a whole.

3) We will operate discussion forums (mailing lists, newsgroups, or whatever seems most appropriate).

4) We will coordinate bug lists, keep track of and publicize works in progress, and generally attempt to provide ``roadmaps'' to the code, and to projects based on the code.

5) And we will, above all, be flexible and responsive. We realize that if we are not perceived as providing a useful service, we will become irrelevant, and someone else will take our place.

6) We are not the primary coders. Most of the code that goes into the distribution will be written elsewhere, both within the Netscape Client Engineering group, and, increasingly, out there on the net, at other companies and other development organizations. (Mozilla "Our Mission")

As one can well imagine, the expectations for the Mozilla project were sky-high across the board, not only within the project itself, at Netscape, but also within the Open Source movement in general and in the media. Eric Raymond's fear that "if Netscape tanks, no one is going to listen to us for another decade"(Leonard "Let my") was not lost on anyone. Mozilla had to produce results, and it had to produce them fast.

After the project had been underway for some time, however, it became apparent that the old Netscape code base, much of which dated back to the original browser from 1994, was in fact causing a number of significant problems. It had become big and bloated over the years and was thus difficult to maintain and hard for outside developers to get a handle on. In late 1998, it was therefore decided to throw out the old code and start over from scratch using a new layout engine named Gecko. The Gecko engine was designed from the ground up to support "open Internet standards such as HTML 4.0, CSS 1 and 2, the W3C Document Object Model, XML 1.0, RDF, and JavaScript" (Mozilla "nglayout"). This was an important design decision that promoted quality, modularity, and

standards compliance, but it came at the expense of the project's ability to deliver a new and improved browser as quickly as most people wanted.

The one-year anniversary of the Mozilla project came and went, and there was no new browser. Outside developers had not flocked to the project as many had hoped. The outlook was pretty bleak, and it didn't become any better when, on April 1st 1999, Jamie Zawinski (in what surely seemed like a bad April fool's joke) suddenly announced that he was quitting the project. In a note posted on his web site he wrote:

> My biggest fear, and part of the reason I stuck it out as long as I have, is that people will look at the failures of mozilla.org as emblematic of open source in general. Let me assure you that whatever problems the Mozilla project is having are not because open source doesn't work. Open source does work, but it is most definitely not a panacea. If there's a cautionary tale here, it is that you can't take a dying project, sprinkle it with the magic pixie dust of ``open source,'' and have everything magically work out. Software is hard. The issues aren't that simple. (Zawinski "Nomo zilla")

Netscape's bold and unorthodox move to go Open Source had ultimately failed to save the company. On March 17th 1999, America Online (AOL) announced that it had acquired Netscape Communications Corporation (Netscape "America Online"). The company that had set out to "change the world" was history, but it had left behind a legacy.

Mozilla.org: A revolutionary project for revolutionary times.

Despite Netscape's demise, the hackers of the Mozilla project continued to work diligently on the open source browser over the next few years. Between March of 1999 and October of 2000, a total of 18 so-called milestone releases were distributed over the Internet. With each new version, the browser got better and better, and people slowly started believing in Mozilla again (Mozilla "Releases"). The number of outside contributors climbed, and so did the number of users. Then, on June 5th 2002, a day that many had thought would never come, the Mozilla project released Mozilla version 1.0. *Salon.com* called it "Mozilla's

Revenge" (Leonard), others were simply glad that there was once again a viable alternative to Microsoft on the World Wide Web.

Slashdot.org: The Hackers' Lounge on the Web

The perhaps most prominent of the web-based hacker communities of the late 90s was *Slashdot.org* (affectionately known as /.). Rob Malda created the site in September 1997 while he was a student at Hope College in Holland, Michigan. The first few years Malda ran Slashdot out of his own bedroom; but, as he says, "from meager beginnings it grew to be quite a popular net destination" (Malda).

Rob Malda, also known as "Cmdr Taco," is in many ways the quintessential icon for the new breed of hackers of the 1990s. He grew up in a small town where computers and the Internet came to represent his window on the world. He says:

> From the days of TRS-80s and BASIC with line numbers, on out to C, C++, SQL, Pascal, Ada, and pretty much any other language that you might want to mess with. I've done it. From playing with hard drives and dip switches and cutting yourself on the inside of cases, I've done that too. From DOS, to Windows, to NT, to OS/2 and finally to Linux, I've been there, done that. Computers pretty much consume every waking moment of my life. They're my job, my hobby, and my passion. They provide my artistic outlet, a stream of revenue, and a way to communicate with those I care about. (Malda)

Slashdot was not an ordinary web site. It was designed to be both a news source and a discussion forum. To this end, Malda wrote Slash, a collection of

Perl scripts that interfaces an Apache web server front-end and a MySQL database back-end in a way that facilitates rapid generation of dynamic content. Slashdot is run by a group of "editors" who collect stories and tidbits from the Slashdot readership and post them on the site. On any given day, the headlines could run something like *IBM Working on Brain-Rivaling Computer*, *Your Rights Online: DMCA bad for Apple Users*, *Email (As We Know It) Doomed?*, or *BSD: FreeBSD 5.0 Developer Preview #2* (Slashdot, November 19, 2002). When the Slashdot community moves, web masters everywhere tremble in fear and sites that have their URLs posted in Slashdot stories frequently experience what is called the slashdot effect caused by the thousands of surfers trying to follow the same link simultaneously. Lag, severe server loads, and even system crashes often result. Websites that have experienced this are said to have been "slashdotted," or as most Slashdotters prefer to say it, "/.ed."

Readers of Slashdot can make comments on the stories that are posted, and typically there will be long discussion threads where anyone from the "Anonymous Coward" to the most highly profiled member can make their arguments heard. Members who consistently make good and insightful posts accumulate what is called "good Karma." Once a member's Karma reaches a certain level, he or she gets to be a moderator for a certain period of time. Moderators are the ones who determine which posts gets the most exposure and for this reason, Karma is a coveted commodity among slashdotters. In 1999 VA-Linux (Now VA-Software Corporation) attempted to buy Slashdot, but Rob Malda, for fear of losing editorial freedom, rejected their offer. When the company came back with a new offer in 2000, a deal was made. Malda, apparently, was no longer worried about their intentions. "VA has done a lot of good for the community," he told a reporter for *Salon.com*, "they won't jeopardize

that by screwing with Slashdot -- they have a lot to lose" (Leonard "The Shape"). Whereas Slashdot has captured the communicative power of the hacker community, other manifestations of this power emerged, perhaps most notably in the form of spaces in which to 'forge' source itself.

SourceForge.net: The World's Premier Hacker Workshop

With its 51,000 Open Source projects and more than 515,000 developers, SourceForge.net is the world's premier Open Source workshop (SourceForge, November 19, 2002). The site, created by VA Software and launched in January of 2000, was designed to "enrich the Open Source community by providing a centralized place for Open Source developers to control and manage Open Source software development" (SourceForge "About"). SourceForge is also a formidable recruiting agency for the Open Source movement in that it also serves as a place where people who are interested in becoming involved in Open Source development can find projects that interest them.

Most independent Open Source programmers typically do not have access to the logistics and infrastructure that commercial software companies do. SourceForge, therefore, helps level the playing field by offering important support such as CVS repositories, home pages, mailing lists and newsletters, bug report and tracking systems, compile farms where developers can test their software on different platforms, and much more. In essence, SourceForge acts as a clearinghouse for ideas and innovation on a scale that has hardly been seen in the software business before. In addition to Slashdot's forum for communication and SourceForge's clearinghouse of productivity, the other key factor in open source development on the web centers on distribution, on download sites such as Freshmeat.net.

Freshmeat.net: The Hacker's Supermarket

The perhaps most popular distribution site for Free- and Open Source software at the turn of the century was Freshmeat.net. Like Slashdot and SourceForge, Freshmeat.net was operated under the umbrella of the Open Source Development Network (OSDN), a wholly owned subsidiary of VA Software Corporation. At the time of this writing (2003), the site maintains an index of over 20 different software categories comprising more than 35,000 software titles (Freshmeat "Welcome").

The main thing that differentiates Freshmeat from other software download sites on the web is that most of the software is licensed under the GNU GPL (71% of all projects) or other OSD-compliant licenses, which means that users have full access to the source code of the programs they download. The site is not just a Mecca for download-happy hackers, however; they also maintain an archive of articles and editorials pertaining to Free- and Open Source software development, an email newsletter with information on the latest and greatest releases, as well an IRC channel for chat and informal communication.

Worldwide Hacker Communities

When Steven Levy wrote about the hackers of the 1960s and 70s, he was talking about local hacker groups who really didn't have much contact with one another or a common sense of community. The Internet changed all that. Communication technologies such as Email, Usenet, IRC, and the World Wide Web brought hackers together in communities that spanned the globe and fostered a self-

conscious movement that made significant and lasting impacts upon technology, society, culture, and economy.

In the 1980s and early 90s these communities were typically found on electronic bulletin boards (BBS), in Usenet news groups such as *comp.os.minix* and *comp.os.linux*, on email discussion lists, or in online chat environments such as IRC. With the advent of the World Wide Web and the increasing amount of spam that eventually befell Usenet, however, from about 1995 onwards the serious hacker communities gradually shifted to the web. Slashdot.org, SourceForge.net and Freshmeat.net are all important arenas where hackers congregate today. Each represents different but important community functions. Slashdot.org is like a giant lounge where hackers and others who are interested in what goes on in the hacker movement gather to get news and exchange opinions. SourceForge.net is the workshop where hackers get together for collaborative projects, learn from each other, and practice what they all love to do. Freshmeat.net is the hacker's supermarket, and along its virtual aisles there is something for everyone.

Each Open Source project is a community in itself. Apache and Mozilla, discussed earlier in this chapter, can be viewed as technical communities created for specific purposes. They consist of a core group of developers with a wide circle of users around them. These communities work toward the specific goal of creating technologies. Slashdot.org, SourceForge.net, Freshmeat.net and a number of other similar communities serve a different but equally important purpose. They constitute the social and cultural sphere that encircles all the technical communities and help give the hacker movement as a whole a common understanding of itself and its mission.

Conclusions

During the 1990s, the hacker movement has evolved from what many would consider an obscure underground phenomenon into something of a mainstream mass movement. The main driving force behind this astonishing evolution was primarily the success of the Linux operating system and the new Bazaar-style development model that followed in its wake. One could put forward a very convincing argument that most of the early projects and developments in the 1990s owed their life to the "collaborative climate" created by the Free Software movement. Richard Stallman's tireless efforts to spread the gospel of Free Software, the GNU General Purpose License, and the free software programs of the Free Software Foundation are all important preconditions for the developments that followed. The split that occurred when the Open Source Initiative entered the stage in the late 90s was perhaps unavoidable, and in the grand scheme of things, something that ultimately, I believe, strengthened the movement as a whole. While the Free Software wing headed by Richard Stallman continued to be a social movement with a carefully thought out philosophy on software development, the new Open Source wing was for the first time able to rally the support of the corporate world. Many important players such as Apple Computer, IBM, and Sun Microsystems adopted open source methodologies as part of their product development strategies, and although many were yet to be convinced, at the turn of the century proponents of both Free Software and Open Source had every reason to be bullish about the future.

5

From Code to Community
MUD and the Case of LambdaMOO

You open the closet door and leave the darkness for the living room,
closing the door behind you so as not to wake the sleeping people inside
—LambdaMOO Exit Message

Every once in a while, new and uniquely different technologies emerge. The hacker movement has produced many such technologies over the years, and the case study in this chapter focuses on one: the Multi-User Dungeon, or MUD for short. It is not meant to be an exhaustive inquiry into the history of MUDs. Rather, my aim is to explore and highlight the dynamics of user-driven technological development as seen in the case of MUD. MUD is a good example of a hacker technology through and through. It was developed exclusively by its users and for no other reasons than that it posed fun and interesting challenges. Although commercial motives played a role in the early MUD development, almost all the significant developments since have happened through extensive code sharing in the MUD hacker community.

I have chosen to follow one particular branch of the MUD evolution. This branch is a system known as MOO (Multi-User Dungeon, Object-oriented). The

chapter opens with an account of the early days of MUD and goes on to follow the specific branch of MUD technology that produced the MOO system. The main part of the chapter discusses MOO developments at a specific site named LambdaMOO. One of the most interesting aspects of the LambdaMOO development is that it took place inside the social and cultural setting of a MUD community. Because of this, the story of LambdaMOO serves not only to show the technological evolution of a unique hacker technology, but also shows how technological solutions were forged in a collaborative effort to meet the social and cultural challenges of the community in which it was situated.

A Short History of MUD

The history of MUDs began in 1978 with Richard Allan Bartle and Rob Trubshaw, then undergraduate students at Essex University in the United Kingdom. Like many of their fellow college students at the time, they were both thoroughly fascinated by role-playing games such as the highly popular Dungeons and Dragons (D&D). In these role-playing games players assume the roles of fantastic mythical characters such as elves, dwarves, paladins, and sorcerers and enter into a Tolkienesque world of adventure and fantasy. Dungeons and Dragons, or D&D for short, was a traditional manual game where players sat down face to face with detailed rulebooks, pen and paper, and several fanciful many-sided dice. One of the main attractions of the game was the social experience of the gaming situation. To successfully solve an adventure, the players had to work together as a team and this created a collaborative atmosphere that was quite different from the typical competitive nature of most other games. Another important reason for its success was its almost infinite replayability, thanks to a rapidly growing number of commercial add-on stories.

In addition, especially creative players could also write their own adventures and invite their friends to play using the D&D rules.

Bartle and Trubshaw had also seen the popular computer-based adventure game, "Adventure." This game is generally considered the first of its kind, and it was created by William Crowther in 1972. In addition to being a computer programmer, Crowther was also an avid cave surveyor, and when he decided to write a computer-based adventure game for his children, setting it in a world of caverns and dungeons was perhaps an obvious choice, especially since Crowther was also a fan of role playing games such as D&D. Crowther's game soon became very popular and was frequently shared among friends and passed along over the computer networks. In 1976 development of "Adventure" was taken over by Don Woods, and much inspired by J.R.R. Tolkien's *Lord of the Rings*, he added many of the fantasy features and game elements that became the hallmark of the computer adventure game genre. Also in 1976, Jim Gillogly at the Rand Corporation ported the original FORTRAN code over to C so it could run on Unix systems, and as a result the game soon spread like wildfire across the international computer networks (Adams).

For computer science students with a fascination for role-playing games, the idea of combining the role-playing and multi-player aspects of D&D with the computer-based structure of "Adventure" was not a strange one for Bartle and Trubshaw. In fact, it was precisely the sort of challenge that many young hackers would find highly interesting. In his online biography, Richard Bartle recalls:

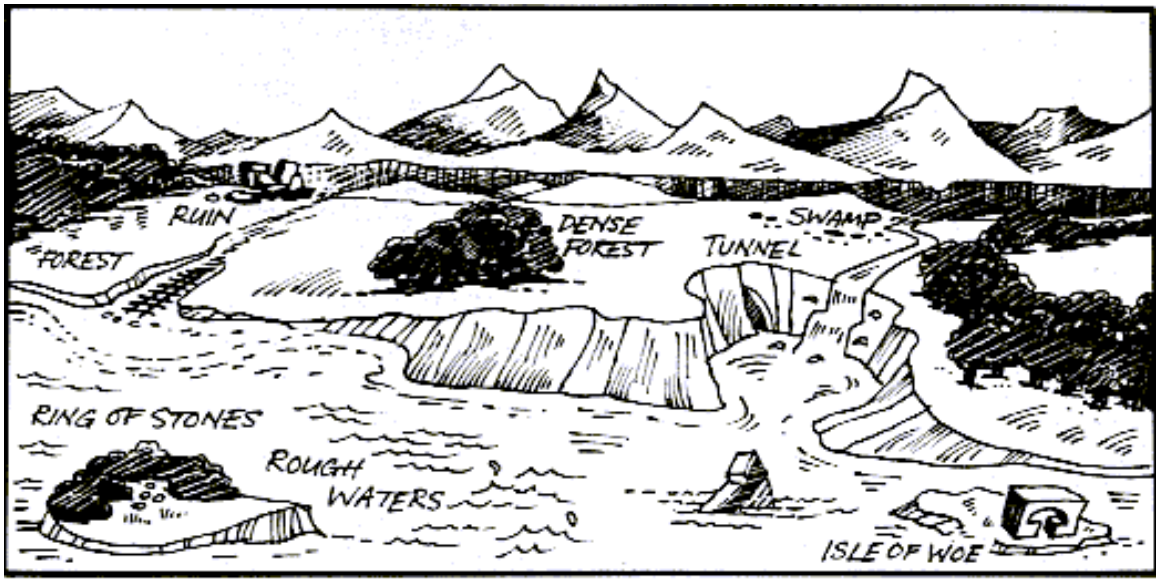
> Trubshaw soon after hit upon the idea of writing a similar game which would allow several people to share the game world at the same time. He

> called his creation MUD and devoted the remainder of his undergraduate career to writing it. (Bartle "Biography")

Toward the end of 1979, Trubshaw had written two versions of MUD. They were both prototype systems written in assembly and because of this, he felt they had become unwieldy. He eventually decided to rewrite the whole system using the BCPL programming language. Bartle, who had done some scenographic design and contributed ideas to the first two versions, now joined the programming and development effort in earnest (Bartle "Interview").

> Roy had got the shell of his third version of MUD running by the end of his course, but didn't have time to complete the rest of the game. He passed what there was to Richard, who added perhaps 75% of what constituted the final program. Roy had the "engine" working, but it didn't do much; Richard enhanced it, and employed it to manage a fully-realised game world. (Bartle "Biography")

After he had finished work on the third revision of what would be known as MUD1, Roy Trubshaw graduated and left Essex University. Bartle, who went on to join the university's graduate program in computer science and eventually became a faculty member there, took over the system, and in 1980 he set up the first public MUD on the University's DECsystem-10 computer. This MUD went under the name of Essex MUD and was open to British players via an experimental packet-switching network called EPPS, which connected six universities in the UK. Players from outside the UK could also connect via the ARPANET. Essex MUD was open only at night, between the hours of 2am and

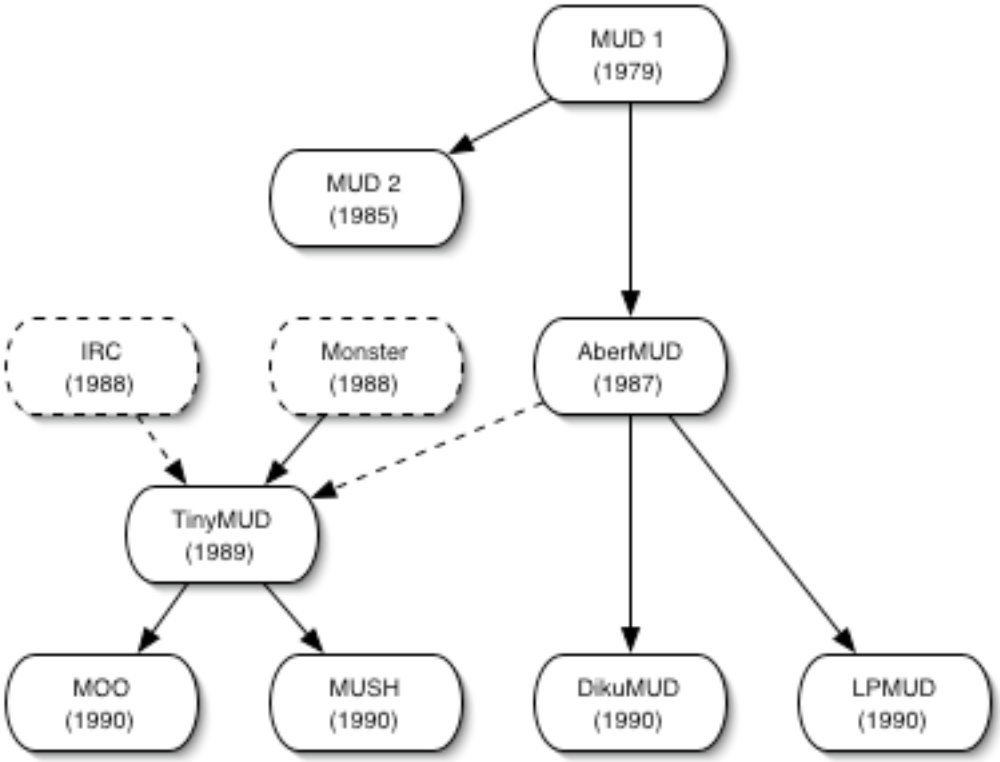


MUD2: View of The Land from Beyond the Vicious Rocks. Courtesy of Richard Bartle.

6am, when no one at the university used the computer for work. It is very likely that these opening hours contributed a great deal to the popular perception of MUD players as nocturnal creatures who spent long countless hours in the solitude of their dorm rooms, hunkered over their keyboards and hacking their way to the fame and glory of wizardhood. By the time Essex MUD "closed its doors" in 1987, Bartle had already developed a new version of the system, dubbed MUD2. The new system had grown out of Bartle's Ph.D. work in artificial intelligence and his desire to improve upon the earlier version of the game. By this time the MUD system was enjoying a good deal of popularity among fantasy and role-playing gamers and to capitalize upon this interest, in 1985, Bartle and Trubshaw formed a company named "MUSE" to market the system commercially. When Richard Bartle eventually left Essex University in 1987 to work full time on "MUSE" and MUD2, the MUD phenomenon was on the verge of its golden age that would soon take it far beyond the imagination of its creators.

In addition to Essex MUD, there was also one other official MUD1 system in operation in the 1980s. This was the now famous "British Legends," which ran on CompuServe from 1987 until 1999 when the company's DECsystem-10 machine on which it ran was finally decommissioned as part of Y2K cleanup efforts (Toth). Thanks to Viktor Toth (a.k.a. MrSpock), in 2000 British Legends was ported to Windows NT/2000 and can still be enjoyed by fantasy players with a flair for history (British Legends). Although, Bartle had, in the early days, let a few other people have copies of MUD to run on their own systems, the commercial interests behind the software did not permit a widespread distribution of source code. However this did not discourage talented young MUD enthusiasts from delving into the breach to design and build their own MUD-type games, and between 1987 and 1991 a host of new MUD systems appeared on the scene. Just like in the case of the original MUD ten years earlier, most all of these second generation MUD hackers were typically college students with a passion for adventure gaming and computer programming.

One of the first and certainly most influential of the second-generation MUD systems was AberMUD. It was written in 1987 by Alan Cox (a.k.a. Anarchy), who was an avid player on Essex MUD while a student at University College of Wales, Aberystwyth. Cox later became a central figure in the Linux kernel development. AberMUD was largely inspired by the original Essex MUD; but it was an independent work done by Cox with the help of a group of other hackers at Aberystwyth. Unlike commercial MUD games such as MUD2, play on AberMUD was free, and it soon attracted a large following among students and hackers both in Europe and the United States. The main influence of AberMUD, however, was due primarily to the fact that its source code was freely distributed and shared on the Internet. This allowed anyone to inspect the inner workings of



The Evolution of MUDs with a focus on TinyMUD.

the system and make changes and new modifications to it as they saw fit. Furthermore, since the AberMUD source code was written in C, it was easily modified and readily portable across Unix systems. As a consequence, AberMUD spawned a flurry of new MUD developments in the late 1980s and early 1990s.

TinyMUD

In the summer of 1989, James Aspnes, a graduate student at Cornell University developed a new type of MUD that he called TinyMUD. Aspnes knew about AberMUD and similar systems, but he had not studied MUD source code as such. He had, however, studied the source code of another popular game called "Monster," and based on this he set about to write a multi-player game that would allow players to collaboratively build the game world and the quests and

puzzles within (Aspnes “Interview”). In the typical hacker fashion, Aspnes had the first version of his system up and running in record time—just a weekend of fast and furious hacking.

> From asp Sat Aug 19 05:47:38 1989
> To: bsy, clamen, nan@helios.ucsc.edu
> Subject: TinyMUD now available via telnet
> Reply-to: asp@cs.cmu.edu
>
> do "telnet LANCELOT.AVALON.CS.CMU.EDU 4201" to connect. The game should be reasonable self-explanatory.
>
> If you have trouble, use the "gripe" command inside TinyMUD to complain (or if you don't, use it to let me know what you think). Don't expect too much out of the parser.
>
> Note: it's ok to build things, the database will (mostly) survive crashes and new program versions.
>
> --Jim (Aspnes “TinyMUD”)

TinyMUD rapidly became a phenomenon that outgrew even its creator’s wildest expectations. People from all over the world began inhabiting the virtual world of TinyMUD, and it gradually took on a life of its own and evolved into something that its designer had not anticipated. The computer program became a community.

An important reason for TinyMUD's almost instant popularity was that many found out about it via highly populated Usenet news groups such as *alt.mud*. Another reason that might help explain the TinyMUD phenomenon is that its creation happened to coincide with the popularization of the synchronous online communication form known as Chat. In the late 1980s, Internet-based programs such as Internet Relay Chat (IRC) had created a new digital space in which communication among people from all corners of the world could take place. The structure of IRC closely resembled its asynchronous counterpart, Usenet, in that communication was centered on topics, or channels to which interested parties could subscribe or connect. In the early 1990s, IRC was growing fast and for millions of people around the world, the idea of talking to other people, most often anonymously, by typing on a computer became second nature akin to chatting with friends on the phone.

TinyMUD, as well as all other MUD systems, could easily facilitate the type of chat found in IRC. In fact, this type of informal communication had been going on in MUDs since the very beginning. What made TinyMUD different, however, was that it had the capacity to create a rich and diverse environment in which to situate the online communication.

In the original MUD, only the game's designers could add to, or modify the game world in significant ways. In MUD 1, for example, Richard Bartle had been the one to design and build most of the spaces and quests in the game. In later systems, such as for example Lars Pensjø's LPmud, dedicated players who achieved wizard status by playing the game were given access to a special programming language known as LPC that they could use to build new areas of the game world, design new quests, monsters, objects and so on. In TinyMUD, however, the power to create could be obtained by anyone. This in itself was a

radical departure from previous systems, but even more groundbreaking was the fact that players performed these building activities directly inside the virtual world itself though the use of special commands like "@dig" and "@create." With these simple but powerful building tools, users were able to construct digital landscapes for a variety of purposes. While many of the spaces in TinyMUD remained game-oriented like Aspnes had envisioned, a growing number of them also served purposes of a purely social nature, or no particular purpose at all.

In TinyMUD, someone who was interested in, for example, 20th century Science Fiction literature could easily create a discussion forum that consisted of several thematic spaces in which to conduct these discussions. Anyone with Telnet and an Internet connection anywhere in the world could easily connect to this space inside TinyMUD and participate in the synchronous discussions, or just hang out and visit, meet new people and make friends in far-flung places. The attraction of TinyMUD was the social interaction that took place in the virtual spaces created by its users.

Although TinyMUD represents a milestone in the history of online multi-user systems, its lifespan remains remarkably short. After only seven months, TinyMUD began to crumble under the weight of its own popularity. Like a frontier town that had grown too quickly without any plan or regulation, the TinyMUD world had, as Aspnes later put it, become "a bloated and poorly-managed slum overshadowed by its more youthful cousins. It was time to put the sorry bold beast out of its misery" (Aspnes "Interview").

MOO

Although TinyMUD Classic was gone, numerous TinyMUD clones continued to draw in more and more people. The MUD phenomenon was just starting to gain

momentum. One hacker who was totally fascinated by Aspnes' TinyMUD was Stephen White (a.k.a. Ghondahrl). In 1990 he wrote, "I've been bitten by the MUD programming bug, a most heinous disease" (White). His interest in MUDs related, among other things, to ways in which they could be used for "writing multi-user interactive fiction and creating a virtual reality." White's first attempt at modifying TinyMUD to his own specifications he called TinyMUCK. He wrote the new system in "one sleepless weekend" using Aspnes' TinyMUD source code and built his own extensions on top of it. Although he thought of it as a "neat hack" at the time, he eventually came to the conclusion that there were several problems with it.

> I started thinking about what TinyMUD is. Okay, so it's like a virtual-reality, whose neat attribute is that you can modify the VR from _inside the database itself_. No need to have separate compilation or external control; you're actually "in there", modifying the way things work. This seemed to be the antithesis of edit-compile-run programming, whose natural interface is "batch mode", rather than interactive. (White)

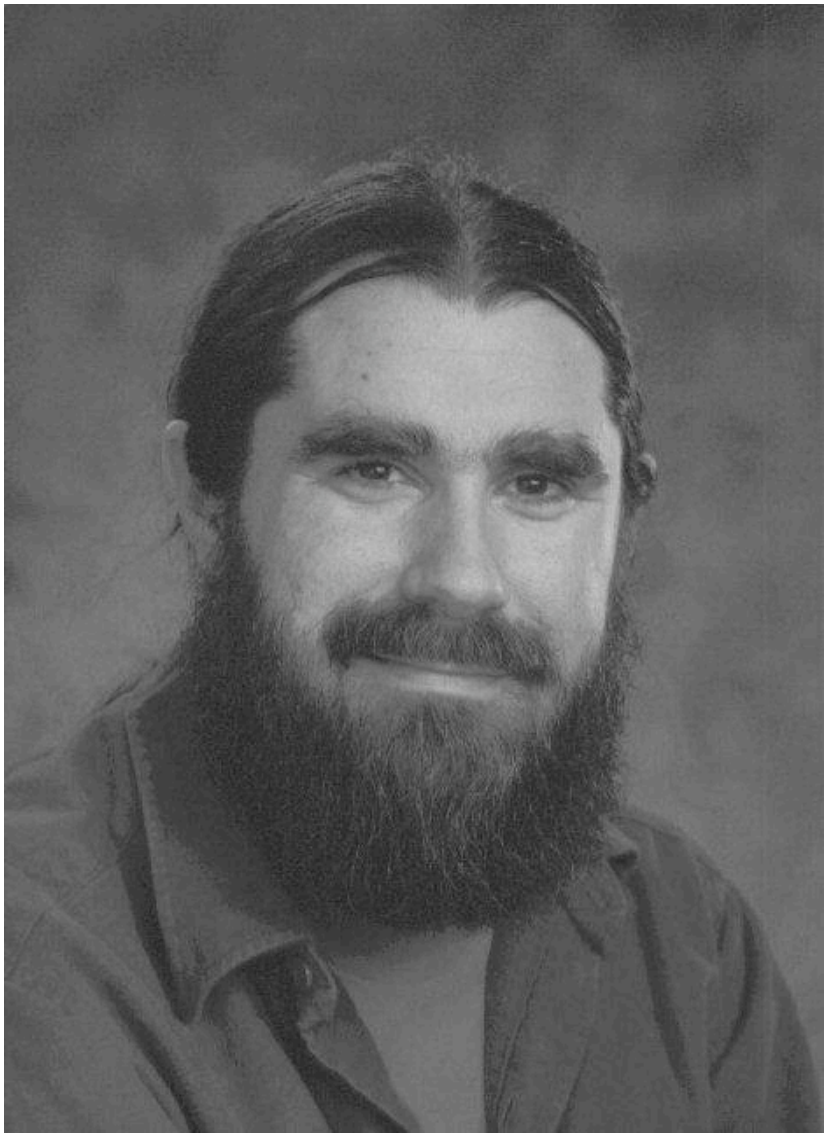
One of the ideas that White had started thinking about was the addition of a programming language to the MUD system. The existence of a powerful tool such as a programming language could, he surmised, make objects a whole lot more interesting and interactive, and the game world as a whole a lot more immersive.

> Okay, so here's a dilemma: my experiments with TinyMUCK seemed to indicate that a programming language was needed, but this seems to

> violate the nature of the VR. Hmm. Well, I talked with James Aspnes about it, and he seemed to be in favour of a programming language. After talking with him, he convinced me that it was the way to go. (White)

White's second attempt at creating a TinyMUD-type system was dubbed MOO, which stood for Multi-user dungeon Object-Oriented. Around 1990, object-oriented programming was fast becoming the new program design paradigm in computer science, and it is not surprising that White would tack on to this new wave spearheaded by widely used programming languages like, for example, C++. With MOO, White says, he "opted out of the 'one crazy weekend' approach, for the better of the code and my own mental health" (White). He spent two months on the design and actively solicited comments and feedback both in Usenet discussion groups and in MUDs that he frequented. MOO was his big project, and he wanted to get it right.

In the summer of 1990, Pavel Curtis, a research scientist at Xerox PARC and a graduate of Cornell University, first stumbled upon MUDs. Curtis had found out about MUDs through MUD-related news groups on Usenet and had looked at some of the systems he could access online. He eventually came across Stephen White's MOO system running on a machine at the University of California at Berkeley. The MUD was called AlphaMOO, and when he logged in, he met a person there by the name of Frand who told him about the system and provided him with some documentation. Curtis studied the information carefully and concluded that the MOO system showed promise and was worth a closer look. Because of his background in programming language design and implementation, he found the MOO language that White had designed of particular interest. He says: "Stephen had actually done a reasonably tasteful



Pavel Curtis creator of LambdaMOO. Photo courtesy of Pavel Curtis.

language and I think that's remarkably rare among amateurs" (Curtis "Interview"). He approached White and suggested that they collaborate on further developments of the MOO system. The idea of collaborating on programming projects was not novel to Curtis. He was at the time familiar with the Free Software Foundation and their GNU project and had even contributed some to that project in the form of bug reports and bug fixes. Also, Xerox PARC had a large code base that was freely shared among its programmers, and he had

frequently made contributions to that. Stephen White, who had been working on MOO on and off for some time, readily agreed and sent Curtis a copy of the MOO source code. Curtis recalls; "It was his [White's] first large C program so it had some of the attributes you can imagine, but overall he had done a pretty good job. He had done a good job of taking code that others had written, the TinyMUD, or a variant of the TinyMUD network module, and he had brought that in" (Curtis "Interview").

Pavel Curtis was different from most other MUD hackers in the sense that not only was he was a professional programmer with a PhD in computer science, but he also happened to work for one of the most respected research institutions in the United States, Xerox PARC. To be sure, his interest in MUD technology was driven by much of the same curiosity and fascination for clever coding so characteristic of hackers, but because of his more extensive experience and training, he was able to look at the MOO project from a systems point of view and apply his knowledge of large complex systems to the MOO project. White had built a promising foundation upon which Curtis could refine and enhance the system. As Curtis poured over his copy of the MOO code during the early fall of 1990, he would occasionally send bug reports and bug fixes back to White. He says:

> I started sending him bug reports. I had fixed my copy [MOO server], and I sent him a disk; but at some point he declared that he wasn't really doing much with it. He said something like, I am not actually preparing a release right now so you go ahead, and that's when I started sort of owning it. (Curtis "Interview")

The Beginnings of LambdaMOO

Curtis now began working on the MOO system in earnest. He had numerous ideas for enhancements and new features that he wanted to incorporate. One of the things he was particularly concerned about was a security issue known as "Buffer overrun." Only a few years before, in November of 1988, Robert T. Morris, a young graduate student at Cornell University, had written a program later known as the Internet Worm, which caused serious havoc at Internet installations all across the world due to its ability to propagate by exploiting security holes such as buffer overruns. Curtis was not only concerned with purely technical aspects of the MOO, however. He had also begun to think about developing a MOO world that would complete the system.

Being new to MUDs, Curtis had assumed that because AlphaMOO was running on a machine at Berkeley, the people he met there were also from Berkeley. When he eventually learned that they all came from totally different places, as far flung as Australia and Israel, it came as a total surprise, and it really opened his eyes to the community capabilities of MOOs (Curtis "Not Just").

The MOO system consists of two main components. The server, which Curtis inherited from White and on which his initial work focused, is what we may call the MOO's "operating system." It handles all the low level system tasks such as networking, communication management within the system and more. The other component is the MOO world itself. This is a database with information about objects in the system, how they relate to one another, how they look, who owns them, and so forth. Whereas the server is mostly transparent and invisible to the users, the database is tangible and concrete. It *is* the virtual world that people experience when they go to a MOO.

On his first explorations into the world of MUDs, Curtis had used the handle *Lambda*, so now, when he had his own system, the name LambdaMOO was a natural choice (Curtis "Not Just"). By the end of October, 1990, Curtis felt that LambdaMOO had progressed to a point where he could start sharing it with other people, so on Halloween he invited two of the people he had met in AlphaMOO to visit the newly established LambdaMOO. These two were Tim Allen, a.k.a. Gemba, and Gary_Severn. They would both play significant roles in the early collaborative developments of LambdaMOO. In Curtis' own words, that particular evening in 1990 came to represent the "first drops of water in what later became the rushing river of LambdaMOO" (Curtis "Not Just" 30).

In the weeks and months that followed, the three of them spent most of their free time working on LambdaMOO. Curtis fondly recalls:

> During the earliest days of LambdaMOO, through the beginning of 1991, everything was fascinating every day. The technical work was especially so as Gary, Gemba, and I tried to build the core libraries of MOO programming, as I furiously wrote new functionality into the MOO server and then furiously wrote new MOO commands that took advantage of it. I pounded the keyboard for hours at a stretch trying to write a comprehensive reference manual for this language and system. (Curtis "Not Just" 30)

The core libraries of MOO programming that Curtis refers to above were the API's (Application Program Interface), a new addition that he had brought into the MOO system. In Stephen White's original MOO server, functions and procedures used by the programming language were hard coded directly into

the server software as built-in functions. The downside of this was that only people with access to the server code could add new programming functionality to the system. Curtis wanted LambdaMOO to promote programming collaboration and code sharing, so he came up with a system of APIs located inside the database itself. These APIs were called Utility Packages, and they functioned as repositories for often-used utility-type MOO programs. With the new API system, programmers could now do all their coding inside the virtual world of LambdaMOO. Curtis says: "The collaborative feel of it was fascinating as we worked closely together from our separate offices thousands of miles apart" (Curtis "Not Just" 30).

From Computer Program to Community

During the early days of LambdaMOO, those who visited came mostly on invitation from Pavel Curtis himself. As it happened, the AlphaMOO at Berkeley had by this time been shut down, and soon people from there, among them Frand, the programmer who had given Curtis his first introduction to MOO, began migrating over to LambdaMOO. A few of Curtis' coworkers at PARC also became interested in the new system, and little by little LambdaMOO began to grow into a small community.

Judy Anderson, a.k.a. Yduj, a legendary LambdaMOO hacker and wizard, recalls her first visit to LambdaMOO.

> This was in early November of 1990. So, I signed on to this thing and started wandering around in it and I said; "Oh, it's a MUD!" He [Curtis] was all disappointed because I had seen one before. I was walking around but he hadn't told me it was the house yet, so it was kind of fun because

> there I was sitting in the house on some computer walking around in the LambdaMOO house, and eventually I said, "Wait! This is his house." (Anderson)

Curtis had built the LambdaMOO world around the physical layout of his own house. In keeping with the informal tradition of MUDs, visitors to LambdaMOO would enter the virtual world though a dark coat closet and suddenly find themselves in the middle of Pavel Curtis' living room. Doorways led off to other parts of the mansion and the surrounding grounds. The textual descriptions of the various locations were descriptive and true to life, and visitors really felt that they were present in Curtis' house without actually being there physically.

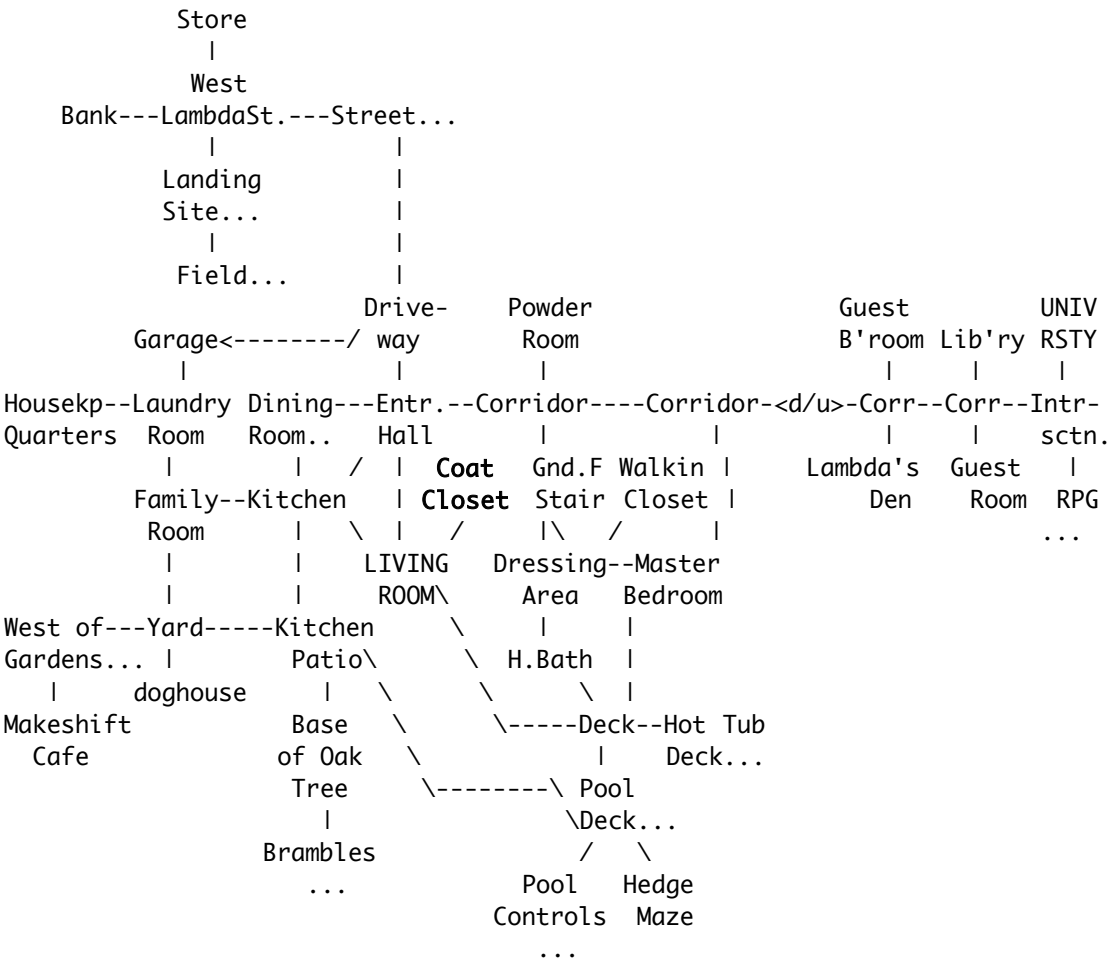
> The Living Room
>
> It is very bright, open, and airy here, with large plate-glass windows looking southward over the pool to the gardens beyond. On the north wall, there is a rough stonework fireplace. The east and west walls are almost completely covered with large, well-stocked bookcases. An exit in the northwest corner leads to the kitchen and, in a more northerly direction, to the entrance hall. The door into the coat closet is at the north end of the east wall, and at the south end is a sliding glass door leading out onto a wooden deck. There are two sets of couches, one clustered around the fireplace and one with a view out the windows.

> You see Welcome Poster, a fireplace, the living room couch, Cockatoo, Helpful Person Finder, lag meter, a map of LambdaHouse, and The Birthday Machine here. Dott (dozing), Jethromeo, Mack-the-Knife (out on his feet), Loki (passed out on the floor), Bear(tm) (distracted), Neeners (out on her feet), GumNut (dozing), Goat, habibi (navel-gazing), Pink_Guest, Subwoofer (dozing), Ultraviolet_Guest, and Beige_Guest are here. (LambdaMOO)

Whether it was the welcoming atmosphere or the exciting technical developments that went on there, LambdaMOO soon became a regular hangout for a growing number of outsiders, and with them came new challenges.

> I used to personally welcome each and every new user for the first few months. […] It was neat, all these people from around the country and, as time went on, from around the world, coming to this program, this server, this place that I had created; they came, many of them stayed, and many of those helped make the place even better. It was magical. (Curtis "Not Just" 31)

Whereas TinyMUD had been an experimental sprawl where pretty much anything would pass muster, LambdaMOO was from the very beginning set on a different path. Curtis had realized that he would need help from other people, not only in regard to the technical programming projects, but as would become abundantly clear later, also with the day-to-day operation of the MOO. To aid him, he therefore invited a few of his most trusted online friends to join in a



ASCII Map of LambdaMOO. Object #15021. May, 13, 2001.

"wizard team." MOO administrators are called wizards because they are the most powerful players in the virtual world. In MUDs, the title wizard was used to denote the highest attainable player class, and was usually only bestowed upon the most dedicated and experienced players. In a MOO, however, a wizard is typically someone who has more in common with a computer systems administrator. Yudj, who had been a close friend and housemate of Curtis in the 1980s, and who had been invited to join the wizard team right from the beginning explains:

> Mostly I was there to get people out of trouble and to help Pavel build up the infrastructure. There was a lot of stuff that we didn't have yet that we have now, stuff that needs wizard permissions in order to run. But there weren't really any duties, I was just logged on a lot and I was spending a lot of time logged on to the MUD rather than working which was a personal problem for me, but LambdaMOO benefited from it. So I just hung out at the MUD a lot and when there was some trouble I tried to help out. I guess back then we had the "Wizards Grant Quota" system, so if people wanted more quota, I helped evaluate whether they should get it and that kind of stuff. (Anderson)

Unlike many other MUD operators at the time, the LambdaMOO wizards were mostly adults. Most of them were also professional programmers rather than amateur hackers, and this gave the MOO a distinct flavor that set it apart from other MUDs. LambdaMOO and its wizards would soon enough experience its share of the social and technical problems that had plagued TinyMUD and other early online communities, but in those early days the MOO was a blissful environment where, as Curtis later put it, "everything was fascinating every day" (Curtis "Not Just" 30).

Throughout 1991 Curtis and his collaborators built most of the core technology that LambdaMOO has been based on ever since. All the technical development, except work on the server, which was mainly done by Pavel Curtis himself, took place inside the LambdaMOO world. It did not follow any set plan or specification. Mostly people would code things that they needed to accomplish some larger project, or things that they just thought would be fun or interesting to do. Programming had been perhaps the most powerful new feature

that Stephen White had conceived of, and in LambdaMOO it was used to its fullest. Yduj says; "The beauty of LambdaMOO is that the extensions, the building, the richness comes from the programming, so we wanted to encourage people to program. So, pretty much if you wanted a programmer bit you could have it" (Anderson). Because of this rich and creative programming activity, LambdaMOO could soon boast a wide variety of interesting objects, puzzles, and components.

LambdaMOO's first puzzle, for instance, was written by Yduj herself. She had designed the master bedroom and adjoining rooms of the LambdaMOO mansion, and to make it more interesting, she programmed a burglar alarm that would be triggered whenever someone entered. The puzzle would challenge the player to find out how to turn the alarm off. Another of Yduj's puzzles, one that I came to experience firsthand on one of my early adventures into LambdaMOO, was a paper bag that, once you entered it, would not let you out until you had written a little program that could disable the bag's locking mechanism. Inside the paper bag there were clues as to how this escape program might work, but it took several attempts to actually get it to open the bag so you could get back out.

> Paper Bag
> Welcome to the Paper Bag puzzle! Hope you're up to it!
> A rumpled brown paper bag with the words "Can you program your way out of a paper bag?" neatly typeset on the side.
> Hyperpelosity (asleep) is here. (LambdaMOO)

The Paper Bag puzzle (object #6231) was not only fun and challenging, but it required you to learn some MOO programming in the process. When asked how she came up with the idea of the infamous paper bag, Yduj says:

> When I make an error [in programming], I often say, "I can't program my way out of a paper bag" as some sort of a self-deprecation way of saying that I made a boo-boo. So, I had this sort of vision that "Look! I have a paper bag that you can program your way out of." So that phrase was the impetus for the creation of that. People are always trying to make little traps and stuff for others, especially to trap guests or their enemies or whatever, anyway, so I made this trap, right, but it had hints and clues as to how to get out. When you try to exit, a verb gets called on your object and it drags you back, that's the mechanism that it uses. It evaluates how it was called and if it was called by a verb that you wrote then it lets you out, if it was called by a verb that you didn't write, like @move for example, then it drags you back. It gives you ten tries or something so if you keep trying it will let you out. (Anderson)

Another example of early creative programming projects in LambdaMOO is the role-playing system. Unlike most MUDs, MOO did not have any role-playing capabilities, so it was not a game in the traditional sense. A typical MUD command such as *kill*, or fantasy player classes such as Elves, Dwarves, and Orcs, for instance, did not even exist in MOO. Although one might play a role-playing game without these features, the lack of a score system as well as a good NPC (non-player character) control system did not really allow for proper role-playing. Gemba and Gary_Severn, who were both dedicated role players, made

it, therefore, their first project in LambdaMOO to design and implement a proper role-playing system. The fact that they could even accomplish a task of such magnitude by using only the MOO language and the programming tools inside the LambdaMOO environment attests to the strengths and versatility of LambdaMOO as a development environment.

In time, many other components would be built, either by the wizards, or by other programmers in the MOO. The in-MOO mail and programming editors, for example, nicely illustrate the collaborative and user-driven evolution of MOO development. Early on, Pavel Curtis had written a simple mail editor that people could use to send electronic mail to one another inside LambdaMOO. There was at the time, however, no programming or verb editor, and this made programming quite difficult for people using client programs such as telnet without cut and paste capabilities. If they wanted to write a MOO program, they basically had to type in each and every line of their code, and if the compiler produced an error, they had to fix it and retype the program again and again until their code compiled. Needless to say this was an arduous process that could dishearten all but the most dedicated hacker. Yduj decided to do something about this, and so she set about to write a verb editor. She says: "I felt so bad for them because they had to enter their whole verb again each time they wanted to make a little change, and I was like, "Oh my God I can't stand this!" so I wrote simpler version of the mail editor" (Anderson).

Yduj's editor was line-based, meaning it would accept input one line at a time just like the default UNIX editor *vi*. Programmers could now open and edit code in the MOO directly. Shortly after she had finished the verb editor, another player, whose name was Roger, told her that he had some ideas for improvements. He had observed that the mail editor and the verb editor

performed basically the same function, namely, allow people to enter and edit text. The fact that one had to do with mail and the other with programming was irrelevant. In an object-oriented system like MOO, abstraction is fundamental. Since both editors performed almost, but not quite, the same function, he surmised, one should be able to combine them. In short, Roger's idea was to build a generic editor that would contain all the code for general editing, and then build sub-classes with code specific to mailing and verb programming. Roger was at the time in the middle of doing his dissertation, but the MOO programming project must have been much more appealing because over the next few weeks he designed and implemented a new generic editor system which replaced both the mail and verb editors.

> It seems that the voices of work avoidance tend to get LambdaMOO having richer stuff. I sort of feel bad that we're kind of a parasites site in that way, but oh well! Anyway, so he wrote the generic editor improving the verb editor that I had written…replaced it and the mail editor that Pavel had written with his new generic editor stuff. That was, I would say in the summer of 91. (Anderson)

Another project that Yduj and Roger worked on illustrates what real-time collaboration on programming in the MOO was like. The project was the MOO's player name database. Yduj explains:

> We had two windows open, we had our non-wizard characters and they were somewhere together talking and we had our wizard characters and they were in private rooms editing. I would say, "I wanna do this to this

> verb," and he'd say "Okay, it's yours," and would work on it, because there is no locking, right, and if we are both editing the same verb, whoever writes it second wins so we had to actually say "I'm going to edit this verb," "Okay I'm done now" so we would not step on each other's toes. That was kind of fun. (Anderson)

This type of close real-time collaboration didn't happen as often as one might think, though. Most of the time, collaborative projects tended to follow the hacker tradition where someone would write a piece of code and then later someone else would pick it up and improve or extend it in some way.

One reason for this must be attributed to the MOO's permission system, which only allows the owner of an object to edit it. In any multi-user system, an access control mechanism needs to be in place to ensure that only those with the proper permissions can read or modify files. In MOO, only the owner of an object has permission to edit it, so if two or more people wanted to collaborate on a programming project, they could not easily edit verbs and properties on the same object without juggling ownership of the object between them. The MOO permission system does have a *write-flag*, which, when set, will allow anyone to edit an object, but the potential security risks involved in this made it impractical and more or less useless for collaborative purposes. The permission control system in MOO does not apply to wizards, which explains how Yduj and Roger were able to collaborate in real-time on the LambdaMOO player database.

Socio-technical Experiences of LambdaMOO

As LambdaMOO's code base grew slowly but surely, so did its user community. After a particularly clever April Fool's Day hack in 1992 where the wizard team

had pulled a prank on Curtis by simulating a system-wide fire and furthermore appeared to be handing out wizard privileges to everyone so they could help clean up the place, the popular press became interested in LambdaMOO. From that point on, the influx of new users would soon number in the thousands, and with them came new challenges of a socio-technical nature that Curtis and his hackers had not foreseen. One such problem arose from the way the character request system was set up. In the early days, character accounts could be created automatically by anyone, and there was no limit on how many accounts a person could have. The problem with this was that troublemakers could not really be banished or shut out from the MOO, since they could easily create another character for themselves and continue making trouble. To deal with this problem, the wizards implemented a character application and registration system where users would be identified by their email address. The development of the player registration database mentioned above was part of this effort. Another issue that became more and more of a problem was intentional or unintentional "theft." A MOO object that is not locked can be picked up by anyone, and more often than not, the person that picked it up would not return the object to its original location. After Frand complained to him about this, Curtis wrote a brief statement called *help manners* in which he outlined what he considered to be appropriate behavior in LambdaMOO. This document was for a long time "the only written 'law' that LambdaMOO had" (Curtis "Not Just" 33). For the wizards, the law turned out to be a double edged sword, because once it existed, someone had to sit in judgment, apply the law and settle conflicts that arose when players broke the law. That someone could only come from the wizards' own ranks, and even if they shared the responsibility, it soon added a considerable amount of work to their burdens. After some time with growing

tensions within the wizard team, Curtis, therefore, decided that the wizards, who in his opinion were hackers and not social arbiters, should no longer perform these duties. In a position statement named "LambdaMOO Takes Another Direction" (LTAND), he outlined a new policy where the wizards would no longer interfere with social life of the LambdaMOO community.

> It was so simple in my mind: there was a clear distinction between social and technical, between policy and maintenance, between politics and engineering. We wizards would become technicians and leave the moral, political and social rudder to "the people." […] In hindsight, I forced a transition in LambdaMOO government, from wizardocracy to anarchy. (Curtis "Not Just" 38)

What Curtis failed to recognize at this point was that there is no clear distinction between the social and the technical, that they are in fact tightly interwoven in a complex mesh that cannot be cleanly disentangled. LambdaMOO would pay for this oversight. Once central control of the community had been disbanded, social conflicts soon reached new heights that culminated with the now infamous "Rape in Cyberspace" affair chronicled by Julian Dibble in the *Village Voice* (Dibble).

The whole affair started when a character named Mr. Bungle created a remote-controlled puppet and used it to verbally abuse other players. The abuse, which had clear sexual overtones, hence the reference to rape, caused a public outcry in the LambdaMOO community and people started calling for the removal of the Mr. Bungle character. Pavel Curtis was away on a business trip when all this happened, and in order to contain the situation, another wizard

took it upon himself to disable Mr. Bungle's account. This reaction, however, fueled another public outcry of even bigger proportions; the wizards had promised to stay out of social conflicts and now they had broken that promise. The incident has been described and analyzed many places (see, for example Vitanza), so I won't go into further details here; but it had a profound bearing on the socio-technical developments that followed. The state of anarchy in which the cyber rape incident occurred had to be brought to an end. Curtis' problem was how to do it without resorting to what he called the "wizardocracy" of past. The solution that he chose was a form of direct democracy using a petitions-and ballots system. Curtis explains:

> Once again, […] I forced a governmental transition on the MOO; we would, I declared, move from anarchy to democracy. […] It was simple in outline: any LambdaMOO citizen could create a petition proposing that the wizards take some action; if it got enough signatures, it became a public ballot measure that passed on a two-thirds majority vote. (Curtis "Not Just" 39)

The technical infrastructure for the new democratic system that the wizards eventually developed is a striking example of the way technological solutions arise out of non-technical challenges and how they in turn are used to support social and political decisions and processes. Through the experiment with direct democracy, the LambdaMOO community was able to pass several successful measures, but according to Curtis, overall it failed to live up to expectations. He says:

> It seems to me now that the voting population could never agree on any measures of real substance. […] On LambdaMOO, this incapacity engendered a profound stagnation; true progress is impossible to achieve in the petition system. (Curtis, "Not Just" 40)

By 1996, social tensions and conflicts again reached the breaking point. Curtis explains: "The wizards have been at every turn forced to make social decisions. Every time we made one, it seemed, someone took offense, someone believed that we had done wrong, someone accused us of ulterior motives. […] The result was a constant stream of messages to the wizards full of anger, suspicion, and of stress" (Curtis "Not Just" 41). Once again, Curtis was compelled to step forward and change the course of events, and together with the other wizards, he drafted a third pivotal policy statement that was made public to the LambdaMOO community on May 16, 1996. In it, he says, "we formally reputed my earlier theory of a social/technical dichotomy; we explained how impossible that fiction was and declared our intent to cease apologizing for our failures to make it reality. It was, in a way, a wizardly coup d'etat; out with the old order, in with the new" (Curtis "Not Just" 41). Throughout the entire affair of social tumults in LambdaMOO, the hackers had learned an important lesson—technology is inherently a social affair.

The Lambda Distribution

From the time Curtis first started hacking on the MOO server, it had been his intention to give it back to the MUD community. The source code had originally come from Stephen White, so his employer, Xerox PARC, he says, was not able to place any intellectual property restrictions on it even if they had wanted to. This enabled Curtis to distribute his modifications freely. The first release of the

LambdaMOO server, as Curtis now called it, coincided with the public opening of LambdaMOO itself in 1990.

> From the beginning my intent was to give it back. In fact, as soon as Stephen sort of bowed out I started hacking on the code in earnest, to fix bugs and improve the quality of things and write documentation. LambdaMOO the server and LambdaMOO the community were announced publicly on the same day and that had been the plan the whole way along. I set certain goals for cleaning up the server and its language early on, and one of the goals was that I wanted a complete manual for the language and its libraries and that there were no built-in commands unless it was absolutely necessary. (Curtis "Interview")

Along with the server, Curtis also distributed what he called a Minimal MOO database that people could use to build their own MOOs. The Minimal database was an extraction of the AlphaMOO core by Gemba and Gary_Severn. It defined the first ten or fifteen objects in the database and was in fact what Curtis himself had used when he built the original LambdaMOO database. The Minimal database had most of the core functionality necessary to run a MOO, but it did not have any of the added libraries and features that were being built in LambdaMOO proper. When the decision eventually was made to include the more feature-rich core of LambdaMOO with the server distribution, it did not come at anyone's request. "It was the natural thing to do," says Curtis (Curtis "Interview").

With a powerful, streamlined server and a rich user programmable core, MOO was now among the most potent development environments in the MUD

world. Many of the more technically savvy users of LambdaMOO soon began to look with interest on the Lambda distribution as a jumping-off point for building their own MOO systems, and in time, so would many others.

The LambdaMOO Server

The LambdaMOO server development was almost exclusively handled by Pavel Curtis. He did have a few summer interns at PARC that helped him out on certain things, but mostly he wanted to be in control of the development. LambdaMOO had become quite a prestigious project. It had from the very beginning been his pet project, so of course he cared a great deal about it for that reason, but, as MOO started to become popular, and more and more people downloaded and looked at the code, it became increasingly important that the code met more rigorous standards for pedagogical clarity, efficiency and elegance. Curtis says:

> I was certainly writing code for an audience both in the sense that I knew there were lots of people who were downloading the server and compiling it and running it, but also in the sense that, and I made a very strong point of this to the summer interns that I got, that the code we were writing was going to be read by other people and needed to be of pedagogical quality. (Curtis "Interview")

The fact that other people would be reading his code and not just using the binary server program led to a heightened sense of self-awareness that ultimately produced a better and more durable product. Being a professional programmer, Curtis tended to be quite strict in matters of quality of code in general, but he

says: "I would not have been quite as strict if I had not been going through an audience. There was also a strong sense that these programs were hard to understand. I knew that they were not that complicated and I wanted them to be understandable, so I had almost a pedagogical purpose in mind" (Curtis "Interview").

His work on the LambdaMOO server had, by 1993, led to a new in-house project at PARC named Jupiter. The Jupiter project was based on the LambdaMOO server, and had a graphical user interface and live audio and video capabilities. It had grown out of the experiences with LambdaMOO, and was targeted at professional online collaboration. Work on the Jupiter project had revealed a number of shortcomings in the LambdaMOO system. One of them was the lack of a proper system for error handling. As he worked on Jupiter, Curtis would make notes of the fixes and new features that he implemented, and in time he would roll many of them back into the LambdaMOO server. Almost none of these new changes came about as a result of pressure from the MOO programmer community. Curtis says, "Nobody outside of PARC was knocking down my door saying we need error handling and we need it more structured than the d-bit" (Curtis "Interview").

Early on, Curtis had set up a mailing list for developers called MOO Cows. The list was intended to function as channel of communication between Curtis, the developer, and the growing community of MOO programmers and administrators. According to Curtis, the list failed to generate any substantial contributions of code to the server development effort, but this, he admits, may partly have been his own doing. He explains: "I would occasionally get server patches, or people pointing out a bug and maybe not necessarily giving me the patch. I may very well have projected a 'this is my server, I'm the one who is

qualified to do this' etc. etc., which may have encouraged people to just tell me about problems and not give me solutions" (Curtis "Interview").

By 1996 Curtis was looking to end his involvement with LambdaMOO. The social tumults in the LambdaMOO community had taken its toll, but more than that, after six years of dedicated MOO development he felt it was time for him to move on. Together with a group of colleagues from PARC he had founded a new startup company, PlaceWare, which developed and sold online conferencing systems. Before he left, however, Curtis appointed Eric Ostrom from the early and very influential Jay's House MOO to be the new server maintainer. Ostrom produced one release of the LambdaMOO server before he in turn left the maintenance to Ben Jackson and Jay Carlson, also of Jay's House MOO. The Jay's house people incorporated many new features and performance improvements that they had developed, and with the establishment of the SourceForge Open Source development site they eventually moved the LambdaMOO server project there.

LambdaCore

The development of the LambdaCore database distribution took a different and more collaborative direction. Although Pavel Curtis was the arch-wizard of LambdaMOO, his influence did not have nearly the same effect as it had on the server development. To be sure, the wizards did develop many of the core tools and features, but other programmers in the LambdaMOO community also contributed to it. Curtis explains:

> On the database side people brought it in [code] themselves, and the key people who were innovating were the Jay's House MOO folks, and Judy

> [Yduj] was one of those. She was a wizard at LambdaMOO so when something was developed over there it got brought in directly, but by and large people were doing it themselves inside the MOO. There would occasionally be a single verb wizzified and we would go and look at it and try to understand what it was trying to do. Then people did things like inter-MOO portals and all sorts of things once open network connections were available and mostly they were just doing those things inside the MOO and sometimes asked us to take them over. So I think it was really quite different from the usual open source thing. (Curtis "Interview")

Over time, the LambdaMOO core became a collage of code collaboratively authored by a large number of programmers. Some contributed large system components such as the editors mentioned above; others contributed smaller programs or improvements on existing code. Sometimes the programmers would team up to implement new ideas, most of the time though they would work alone. Communication between them happened either in real-time in the MOO, or via the MOO mailing list system. Yduj says; "It was a very subtle kind of collaboration because there wasn't a hierarchy of people who decided what to do. The project was always sort of free-floating" (Anderson).

There was no set release schedule for when new versions of the LambdaCore would come out. It would simply happen whenever one of the wizards decided that it was time. Yduj, for example, didn't have any plans one Christmas, so she decided it would be a good time to work on a core release. She says; "I would make the announcement and then there would be a frenzy of work for the next month while people added stuff that they wanted to go in the core" (Anderson).

Whenever a piece of code was put in the core, ownership of it was transferred to one of two special characters, Hacker or Wizard, and there would typically be no trace left of who was the original programmer. Occasionally a programmer would include her name in a comment in the code, but in most cases, programmers who contributed code to the LambdaCore remained anonymous to the outside world. For people who used LambdaCore to build new MOOs, the core appeared to be one large collectively-owned code base.

The Legacy of LambdaMOO

While the life span of the seminal TinyMUD was short and frantic, LambdaMOO is still online today (2003) 13 years after its inception. Over the years the community has changed character as its core users have gotten older. These days LambdaMOO is more like the small sleepy home town where people come to visit old friends and remember the legacy of days gone by when the community they grew up in was right there on the digital frontier of Cyberspace.

In my view, the most important legacy of LambdaMOO falls into three categories. One, which has received a lot of press and scholarly attention over the years, has been the experiments with online government. For easy reference we can divide these experiments into the following stages:

- Feb. 1991 – Dec. 1992: "Enlightened dictatorship."
- Dec. 1992 – May 1996: Anarchy.
- May 1996 – present: Direct democracy.

Through the government experiments, the LambdaMOO community has, as I have discussed above, taught us important lessons about the relationship

between social and technical issues in virtual worlds, and the dynamics between system administrators and users.

Another important legacy of LambdaMOO is the Lambda software distribution. Because of it, starting new MOOs became a lot easier; and as a result, the early 1990s saw a growing number of MOOs coming online. Most of these new MOOs were started by people who had been regulars at LambdaMOO, and many of them were simply emulations of the social space of Lambda. A few of the spin-offs, however, managed to break new ground. Jay's House MOO, for example, which was one of the earliest new MOOs, quickly established a reputation for being a hacker's haven. It was created by Jay Carlson and a few of his hacker friends, and many of the technical advances in the MOO world after Lambda came from there. In the beginning, code improvements and new features from Jay's House MOO would find their way into LambdaMOO, but eventually the Jay's House crowd developed their own core database distribution dubbed JHCore. While being completely compatible with LambdaCore, the JHCore sported a number of new features and enhancements that made it, in certain ways, more advanced that the original LambdaCore.

A third legacy of LambdaMOO is the academic and educational MOO. In 1992, Amy Bruckman, a graduate student at MIT, started MediaMOO as an online meeting place for media researchers. MediaMOO became in effect the birthplace for a whole new direction in the evolution and use of MUD technology. By 1995, the utilization of MOO technology for professional and educational purposes was taken up by new MOOs such as BioMOO, AstroVR, CollegeTown, Diversity University, Connections, MOOville, DaMOO and LinguaMOO, to mention just a few. In the following chapter I will look at the

development of one of these educational MOOs, Lingua MOO, and the enCore Open Source Project that followed from it.

Conclusions

The case study in this chapter has focused on a unique hacker technology, one that did not exist prior to being invented by Bartle and Trubshaw in 1979, and one that could not have existed in its current form if it were not for the evolutionary hacker development model. It all began because Bartle and Trubshaw decided they wanted to play fantasy games in a multi-user online environment. No one had told them that MUD would be a neat idea. They simply invented the concept just because it was an interesting challenge that could produce something that would be fun to play with. When the MUD phenomenon caught on, others such as Alan Cox became interested in how it could be modified and expanded, and since the MUD source code was not available due to commercial restrictions, he and his hacker friends developed their own version of MUD. When the AberMUD source code was eventually released, anyone who was interested could look at it and see for themselves how a MUD worked. This spurred a flurry of activity in the MUD community, and new versions of MUD sporting funny and imaginative names such as TinyMUD, LPmud, TinyMUCK, FurryMUCK, DikuMUD, MUSH, MUSE, and MOO appeared in numbers. Many of these systems had the basic gaming functionality of the original MUD, but they were also individually different. In the late 1980s and early 90s new MUD installations such as the ones mentioned here were one-of-a-kind systems. They were unique because of the extensive system level modifications made by the hackers that established them. With LambdaMOO and the LambdaCore distribution in the 1990s, that uniqueness gave way to a

more uniform evolution. The LambdaCore distribution provided a feature-rich platform that made system level hacking less compelling, and with the addition of a simple but powerful programming language, the focus of MOO development shifted toward community building. Other branches of the MUD technology tree evolved in a similar fashion. The LPmud system, for example, also had a programming language named LPC (after its inventor Lars Pensjø) and a core database, called "mudlib," that ensured consistency and compatibility between the many LPmuds that were built during the 1990s.

The evolution of MUD was, in many ways, akin to a relay race. As old programmers tired of development and moved on, new programmers stepped up to the plate to carry the technology forward. Whereas some branches of the MUD technology tree eventually withered and died because no one took a concerted interest in furthering them, others, such as MOO, blossomed and advanced the technology. The difference between the relay race and the MUD evolution, however, is that the latter did not have any particular goal in mind. It went where its users wanted it to go. There was no one owner or entity that determined its history. MUD technology was owned collectively, and developed collaboratively, by the community of people that used it.

6

Between Theory and Practice
Lingua and the enCore Open Source Project

If we accept the idea that all knowledge is socially constructed, then there is no theory outside practice, and no neutral and objective place outside practices.—Gayatri Spivak

The collaborative software development methodologies, which were the subject of previous chapters, form the framework for the enCore Open Source project that will be discussed here. As I have shown through the case studies of BSD, GNU/Linux, Apache, and Mozilla, the Open Source model has yielded some remarkable results for hackers and programmers in what we might call the traditional areas of software engineering. Through the enCore project I wanted to see if this model could also be successfully applied to programming and development efforts situated in the humanities.

The chapter opens with a brief history of the academization of MOO technology in the early 1990s and goes on to discuss the notion of the educational MOO as it materialized from the mid 1990s onward. As an example of one such educational MOO, I will focus on the Lingua MOO project, which was begun by Professor Cynthia Haynes of the University of Texas at Dallas and myself in

January of 1995. Based on experiences from the Lingua MOO project, in 1997 we began the new project called High Wired enCore. The rationale for this project was to spread the use of MOO technology in education through traditional book publishing efforts as well as the distribution of a MOO software package called enCore. The main portion of this chapter is devoted to a discussion of enCore and the Xpress Graphical User Interface that I later developed under the umbrella of an Open Source project. The chapter concludes with some thoughts on results from this project and the Open Source development methodologies that I used.

Amy Bruckman and MediaMOO: Academic MOOs in the Making

LambdaMOO's growing popularity in the early 1990s attracted all sorts of people to the online community. Some came because they wanted a change of pace from the typical adventure role-playing world of MUDs. Others came and stayed because they enjoyed socializing with folks online. Yet others stopped by simply because they were curious to see what synchronous online communities were all about. While most users were content simply to avail themselves of the services provided by Pavel Curtis and the LambdaMOO staff, others discovered that MOO was a powerful community-building technology waiting to be explored. One of those who saw a potential in MOO technology was Amy Bruckman, a graduate student of MIT's Media Lab, who in October of 1992 founded the online community MediaMOO. She says:

> Most MUDs are populated by undergraduates who should be doing their homework. I thought it would be interesting instead to bring together a group of people with a shared intellectual interest: the study of media.

> Ideally, MediaMOO should be like an endless reception for a conference on media studies. (Bruckman "Finding" 17)

The virtual place that Bruckman created became, in effect, the birthplace for much of the academic research and professional activities related to MOO technology in the 1990s. At its most popular (1995), MediaMOO had over 1000 members from 39 countries with several academic groups such as the Techno-Rhetoricians, the Netoric Project and Tuesday Café, all of which utilized MediaMOO for online meetings and other professional .

In 1993, not long after Bruckman founded MediaMOO, Gustavo Glusman, a student of the Weizmann Institute of Science in Israel, founded another early academic MOO called BioMOO. Just like MediaMOO was designed as an online meeting place for media researchers, BioMOO was designed as a virtual meeting place for biologists. According to BioMOO's purpose statement, which echoed the one Bruckman had written for MediaMOO, "BioMOO was a *professional community* of Biology researchers. It is a place to come meet colleagues in Biology studies and related fields and brainstorm, to hold colloquia and conferences, to explore the serious side of this new medium" (Glusman "Purpose"). Over the years, BioMOO played host to a number of academic groups within biology and bioinformatics. The Ecology and Evolution Journal Club, the Neuroscience Journal Club, and other groups used the space for online meetings in the mid 90s.

Although MediaMOO and BioMOO were not the only academic MOOs in the early 1990s, they are both good examples of the professionally-oriented spaces that people sought to create at the time. In the minds of most people, MOOs and MUDs in general were games and not something suited for serious academic

Amy S. Bruckman, creator of MediaMOO

purposes. The creators of these first generation academic MOO spaces, therefore, sought to legitimize the use of the technology by focusing strictly on traditional professional activities such as conferencing and networking. In the case of MediaMOO, the target was media studies; in the case of BioMOO, Biology and Bioinformatics. Another early MOO, AstroVR, was designed for astrophysicists. Such was the nature of the discipline-oriented first generation of academic MOOs. Special purpose statements were drawn up in order to emphasize the professional nature of the systems, and anonymity, a fundamental feature of role-playing games (RPG) and social interaction oriented MUDs, was abolished in

favor of responsibility, accountability, collaboration, and networking among the members.

While LambdaMOO had always had an open-door policy, where anyone could become a member, the early academic MOOs adopted more stringent entry requirements. Only those who in some way could claim a connection to the mission and purpose of the MOO were eligible to join. In the case of MediaMOO, Bruckman explains that she was "loose on the definition of media—writing teachers, computer network administrators, and librarians are all working with forms of media—but strict on the definition of research (Bruckman, "Finding" 19). Michael Day, one of the MediaMOO veterans explains:

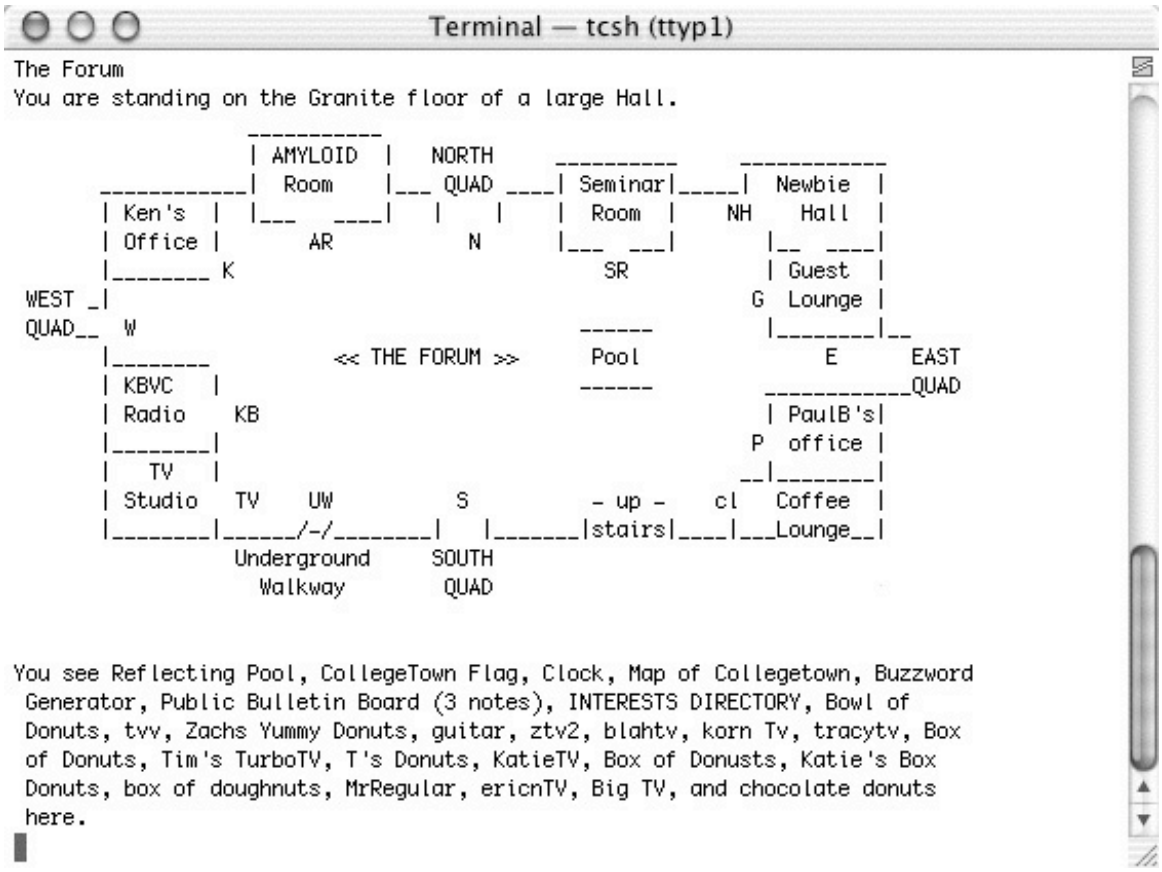
> MediaMOO always has had a strict character application policy. When you do @request name for email, you have to answer a series of questions about your research interests, and at its peak, a team of 5 reviewers would review the applications to let Amy, former head janitor (as wizards on MediaMOO have always been called) know whether to accept someone or not. (Day)

Strict admission policies notwithstanding, MediaMOO was a popular place in the years between 1993 and 1997. Some of the people who found their way to Bruckman's digital world were not content just using the technology for themselves. They wanted to share it with their students to see how such online spaces could help enhance teaching and learning. In MediaMOO, however, there was little or no room for this type of activity. One reason might have been that MediaMOO was so popular and attracted so many users that there was simply no additional bandwidth to accommodate large classes of students. Another

reason, which seems to resonate well with MediaMOO's emphasis on professionalism, might have been that large groups of potentially unruly undergraduate students would not be conducive to the serious academic community that Bruckman wished to foster. Starting around 1994, therefore, a few members of MediaMOO began setting up their own academic MOOs specifically for educational use. The LambdaMOO source code was readily available online, and with some knowledge of MOO and Unix systems it was not too difficult to get started. One of the first new educational MOOs was CollegeTown, built by Professor Ken Schweller of Buena Vista University in Storm Lake, Iowa. Schweller had been a LambdaMOO regular since the early days and also one of MediaMOO's first and most active members. Collegetown MOO was modeled after a traditional college campus with academic buildings, lecture halls and office spaces for faculty situated around the virtual commons. In the small but growing world of educational MOOs, Collegetown MOO was known for its consistent use of detailed ASCII graphics to represent the spatial organization of the digital space.

For many students, and especially teachers in the mid 1990s, the text-based virtual reality of MUDs was something quite foreign and unsettling. A simple two-dimensional ASCII representation of spatial layouts, therefore, went a long way toward visualizing the geography of the textual world. A more detailed discussion of various aspects of spatial representation and layout in MOO design will follow later in this chapter.

Collegetown was only one of many new academic MOOs that sprang up during the mid 1990s. MediaMOO had inspired most of them, but they also had another thing in common, most of them were designed for educational use. In the



Collegetown MOO. ASCII map of the forum.

next section of this chapter I discuss the creation of one of these new generation educational MOOs.

Designing Lingua MOO

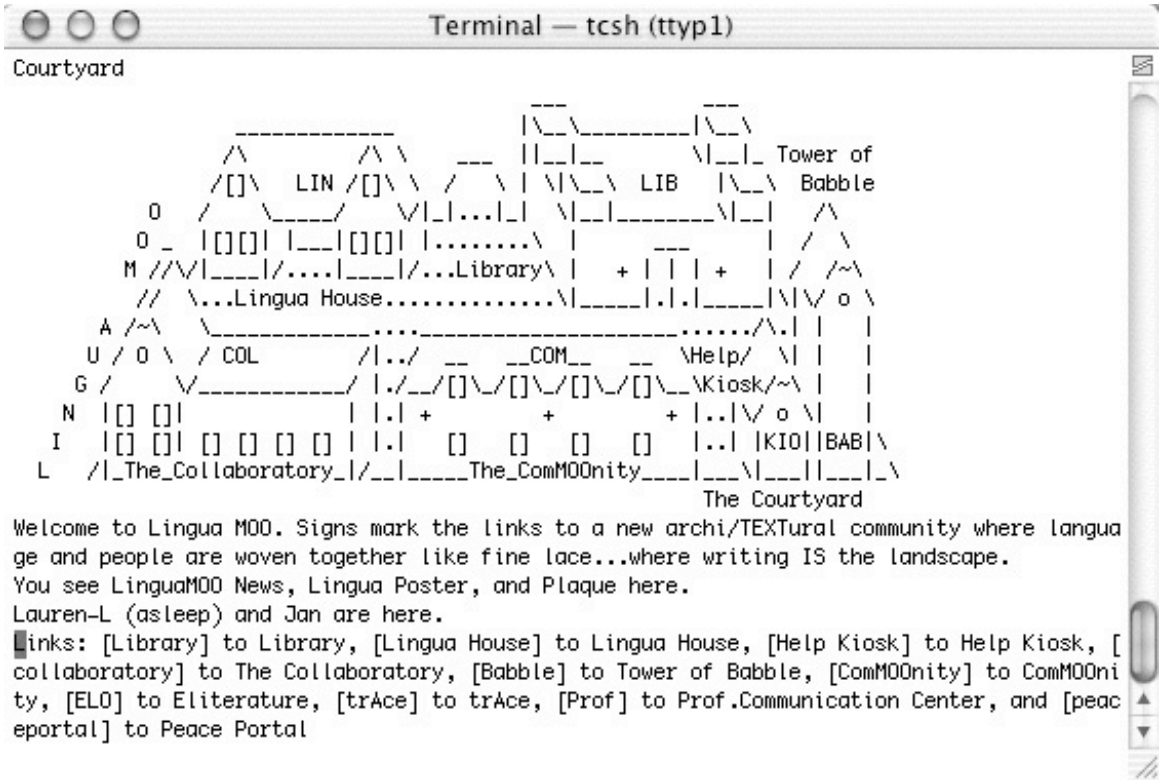
LinguaMOO was started by myself and Professor Cynthia Haynes of the University of Texas at Dallas (UTD) in January of 1995. At the time, Haynes was a visiting professor at UTD, while I was working for a science policy research institute in Oslo, Norway.

When we first started the Lingua MOO project we had two main research objectives in mind. First, we wanted to see whether a technology that had been developed primarily for the purposes of gaming and social interaction could also

prove to be a viable and interesting environment for teaching online. Secondly, we wanted to explore the potential of MOO as a collaborative online research environment. As we set out to design and build LinguaMOO, therefore, we had a set of key design objectives in mind. Specifically, we wanted to create a new type of learning environment that would:

- Facilitate collaboration.
- Encourage communication.
- Stimulate the students' interest in reading and writing.
- Transcend geographical and cultural barriers.
- Be a fun and creative place to work and socialize.
- Provide a space in which to conduct as well as present collaborative research and writing.

As we set about implementing our ideas, the MOO rapidly became the vehicle for its own construction. Working out of her home in Fort Worth, Texas, Haynes drew up plans for the various facilities we wanted the MOO to accommodate, and she secured a space for the project on one of UTD's internet servers. From my office in Oslo I downloaded and installed the LambdaMOO distribution on the UTD server. Once everything was up and running we could both log in from our different locations and begin the task of designing the first few public spaces in the new MOO. Nowadays this kind of tele-work is commonplace, but back then we were both amazed at how well MOO technology facilitated this type of collaboration. It appeared that LinguaMOO was yielding some interesting and productive results even as it was being built.



The courtyard of LinguaMOO. Spatial concepts in textual space.

Before we started actually building the MOO, we went on the Net to see what other people were doing. Besides MediaMOO, BioMOO, Collegetown, and a few others, there weren't many educational MOOs in existence at that time; so we gathered what inspiration we found and added our own ideas about how we thought an online learning space ought to look like. In our original design of LinguaMOO, we wanted to situate the learning environment within the ancient Abbey of Umberto Eco's novel, *The Name of the Rose*. We thought the ancient pastoral setting of a medieval abbey would provide an interesting juxtaposition to the modern day technology used to implement it. Also, by situating the MOO within the framework of a novel we hoped to foster the notion of writing as building blocks of community. Thirdly, the dark, brooding, almost gothic atmosphere of Eco's novel provided a kind of thrill and excitement that we

hoped would stimulate our users to be creative in the design of their own spaces within the MOO.

Choosing the ancient abbey as our architectural metaphor allowed for a star-shaped layout with a central hub, "The Courtyard," and the various main areas of the MOO, such as the teaching and research areas (The Library), the administrative offices (Lingua House), and the player quarters (The ComMOOnity), were directly connected to it. The spatial orientation provided by such concepts as rooms and locations, exits and entrances, objects and players reinforces the sense of space, place, and time in which we wanted to situate the online learning experience. Because these are concepts that we all know well from the world we live in, we hoped they would ease the transition from the physical to the digital learning environment by giving students known concepts and cues with which to navigate, negotiate, and domesticate the space.

During the late winter of 1995 we spent a great deal of time online designing and redesigning the public spaces of Lingua MOO. This included creative work such as building and writing descriptions and ASCII maps for rooms and other objects, but also technical work such as bringing in objects and features from other MOOs, as well as making various modifications and additions to the MOO core database. One of the first things we did in this respect was to remove as much evidence of the system's gaming roots as we possibly could. For example, we replaced the keywords player and wizard with user and administrator. The original words were later reinstated, but in the beginning we felt that it was important to create a professional looking system that would garner the support of faculty members and university administrators.

Although most of the early design work was done by us, we did have outside help on special projects, the most important of which included a simple World

Wide Web interface originally written for Collegetown MOO by Mark Blanchard, as well as a fairly sophisticated inter-MOO communications network by Gustavo Glusman of BioMOO. By late March we felt that the Lingua system was ready to receive its first users. The grand opening ceremony was scheduled for April 4th 1995, and the big day was celebrated with much virtual champagne and fireworks.

Since neither of us had much prior experience with software development and user support, the first couple of months after the opening brought new lessons every day. Among the first things we learned was that in a creative space such as the MOO it is almost impossible to steer and control the evolutionary direction of the space. Enterprising users soon added their own areas to the MOO, which had totally different themes than the one we had envisioned, so over time the evolution of the Lingua MOO environment came to resemble a typical urban sprawl rather than the planned, linear design of a software program.

Another thing that we learned early on was that the users were in fact our most important assets. Not only were they instrumental in helping us find and fix bugs and other malfunctions, but also many of them became important sources of inspiration with regard to ideas for enhancements and new features. A case in point is Dr. Brian Clements (also at that time with UTD) who, besides Haynes, was one of the first teachers to bring students into Lingua MOO. After he had used the system for a couple of weeks, he came to us one day and asked it there was a way to simultaneously monitor and record student activity in multiple rooms. He had already attempted to use the basic recorder object that we had brought in from another MOO but had found it lacking in several respects. We asked him to give us a specification of the features that he wanted,

and with this in hand we were able to develop a new system which became known as the Lingua MOO Recording and Intercom System. Another example of user-driven MOO development is the Moderated Room that we developed for Dene Grigar's doctoral defense, the online portion of which was held in Lingua MOO on July 25th 1995. When Grigar first asked us if she could hold her doctoral defense in Lingua MOO, it was evident that the basic MOO room would not be able to accommodate a serious academic event such as a Ph.D. defense. The main problem was that the basic room had no mechanisms for controlling speech; anyone can say and do whatever they wanted at any time; and this, we surmised, could create problems and reflect poorly not only on the candidate, but also on us and MOO technology in general. We began searching for alternatives and found that Ken Schweller had developed a moderated room for Collegetown that he called the Classroom. This room had a limited set of moderation features in that the owner (typically a teacher) could set up a series of tables at which students could communicate privately without disturbing the rest of the room. With Professor Schweller's permission we ported the Classroom to Lingua and did some testing to see how it would work for a conference type event. After a few weeks it became clear, however, that the classroom's moderation features were not rigorous enough for what we needed. In the process of porting and installing the code for the Collegetown classroom we had learned how the room's speech control system worked; so we decided to build a new moderated room with our own features based on that code. The new room, which became known as the Moderated Room, was based on the notion of an auditorium where the speakers and the audience occupy two different and clearly defined areas of the room. Speakers are often seated on a stage or a panel, whereas the audience members are seated in the audience. This concept allowed us to implement a

system where speakers could address the whole room, but audience members could only talk among themselves. If they wanted to ask a question of a speaker, they had to send it first to a moderator who in turn could field it to a speaker at the appropriate time. To make the room more versatile we also built in a feature that allowed the moderator to open it up for free discussion much like a basic room. As it turned out, Dene Grigar's Ph.D. defense in Lingua MOO became quite a success with more than fifty people in attendance from all over the world (Grigar "Dene Grigar's Online").

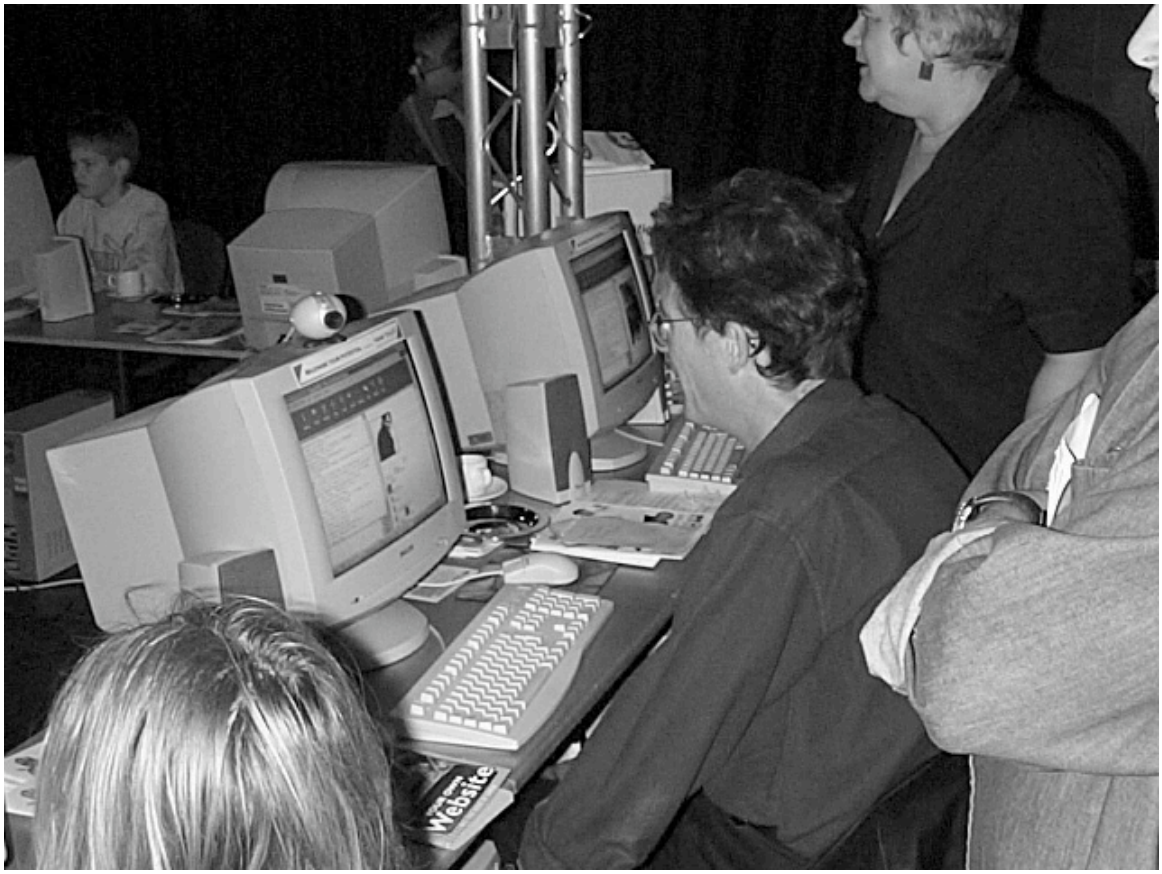
The online dissertation defense, as well as many of the other early Lingua events, demonstrated that MOO technology held a greater potential than many of us had realized. From a technical point of view, what might have taken several professional programmers to write in a traditional proprietary environment, all the technology needed to produce the online defense was accomplished with free software and less than 100 lines of specialized MOO code by two humanities scholars. From a pedagogical point of view, we could see that students and others were able to make the MOOspace their own by constructing personal spaces within the virtual world. The MOO system provides simple but effective tools that allow users to extend and decorate new spaces; and by doing this, we found, users domesticated their learning environment by investing in it. Instead of being simply a tool for learning, like most traditional instructional software, the MOO became a place where they could also engage in extra-curricular activities, such as meeting old and new friends from all over the world, or just hanging out after school. Another thing that became increasingly clear was the importance of the very reasonable technical requirements for accessing and using the MOO system. Basically all that is needed to use a MOO is a simple telnet connection; and this made it possible for anyone with an Internet connection,

regardless of computer platform (i.e., Mac, Windows, Unix, etc.) or network speed, to be able to access the system and use it productively. No student was left behind because of the lack of state-of-the-art technology. In academic settings, where schools often have older computer technology in their labs, this made a strong case for MOO technology in education.

Lingua MOO was primarily designed as a learning environment; and in addition to being used extensively by freshman writing classes from UTD, we also supported classes from other schools across the world. Many of the outside institutions who availed themselves of the services provided by Lingua MOO would later set up their own educational MOOs, while others have continued to take their students to our MOO.

The following list includes some of the institutions that Lingua MOO catered to during this time:

University of Bergen (Norway), Old Dominion University, Hanyang University (South Korea), University of Wisconsin, Shlzuoka University (Japan), Colorado State University-Boulder, Colorado University-Denver, Institute for Media Communication (Germany), Nottingham University (England), George Mason University, University of Southampton (England), University of North Carolina-Greensboro, University of Illinois-Chicago, University of Louisiana-Lafayette, University of Sioux Falls, South Dakota, University of Texas at Austin, University of Texas at Arlington, University of Rochester, New York, Texas Tech University, California State Polytechnic University, Southern Methodist University, Ohio University, Purdue University, Vassar College



William Gibson visits LinguaMOO. October 9, 1999

Over the next few years Lingua MOO also played host to a number of academic conferences and meetings. The organization TOHE (Teaching Online in Higher Education) held their annual conferences in Lingua several years in a row, and other groups such as ELO (Electronic Literature Organization), the e-journal *Kairos*, and *trAce* online writing community also used the space on a regular basis. All of these groups helped make the MOO a vibrant and creative community where you never knew who you might run into.

By 1996 we started to see a growing interest in MOO technology coming from a number of new academic areas. Whenever we presented the Lingua MOO project at conferences such as the Modern Language Association (MLA), Conference on College Composition and Communication (CCCC), and

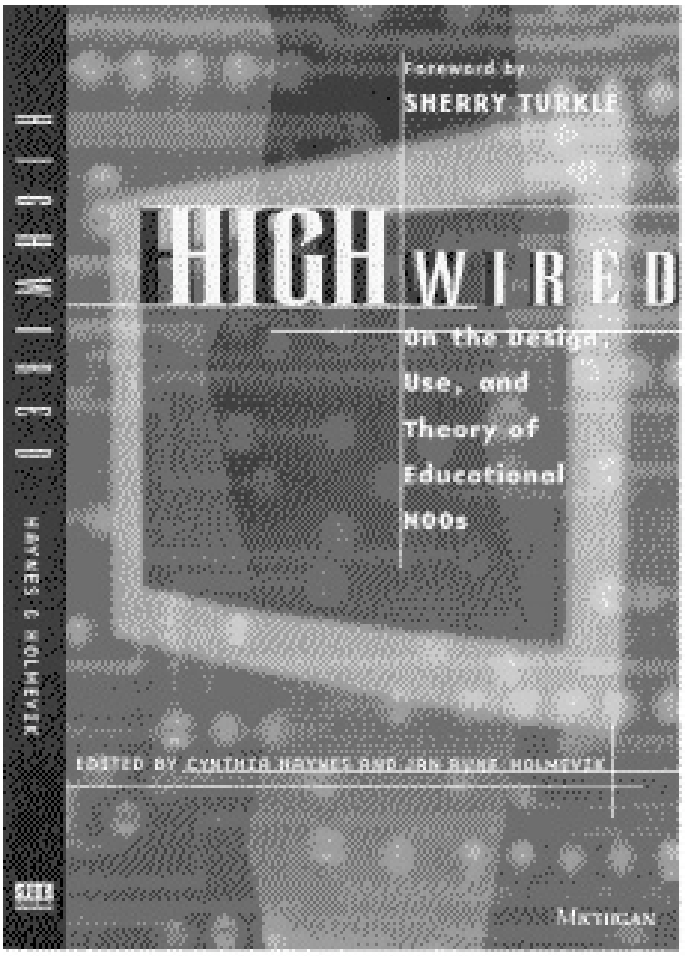
Computers and Writing (CW), there was always a significant interest from people who wanted to learn more about the technology and how it might be used for learning purposes.

From the very start of the Lingua MOO project we had offered educators from other institutions space in the MOO for their own projects and classes. However, as their proficiency and experience with the digital learning environment grew, many of them began to think about setting up their own educational MOOs. Also, during this time we heard from several educators outside the LinguaMOO community who wanted our advice and assistance in starting educational MOOs of their own. It was clear that we could not possibly help everyone who wanted to set up a MOO, but what we could do was to create a resource that would teach interested parties how to establish and teach with MOO. Thus was born the High Wired project.

High Wired: The Beginning of an Open Source Project

While the philosophy of Free Software was fairly well known in most technical areas of academia, in the humanities it was still largely an unknown phenomenon. Thus, it didn't occur to us at first to create a complete software package as part of the effort to get more educators involved with MOO. Instead we went the traditional humanities route of publishing a book.

The ideas for the book had begun to materialize as early as the summer of 1995, and the basic idea was to create a resource that would teach prospective MOO administrators and teachers not only how to set up and run a MOO, but also instruct them on how to teach with it, as well as provide a theoretical framework for educational MOO activity. To this end we invited a number of scholars and researchers with an interest in MOO technology to each write a



High Wired: On the Design, Use, and Theory of Educational MOOs. University of Michigan Press, 1998.

chapter on their respective areas of expertise. As it turned out, the University of Michigan Press picked up the book early on; and for the next year and a half we were busy writing and editing the manuscript of the book which came to be named *High Wired: On the Design, Use, and Theory of Educational MOOs*. The first edition of *High Wired* was published in 1998, while a second edition of the book with updated information on how to set up enCore MOOs with a new graphical user interface named Xpress was published in 2002.

Our original idea had been to base the technical references in the book on the popular LambdaCore distribution; but in the late spring of 1997, as we were

putting the final touches on the *High Wired* manuscript, we came to realize that it might be helpful to have a special education-oriented version of LambdaCore to go with the book. We knew from experience that the first task that faced a new MOO administrator was to port and install popular educational tools such as Ken Schweller's Classroom object, our own Moderated room, recording systems and more. It seemed to us that to provide a special MOO core with all these tools pre-installed would save new MOO administrators a lot of time and trouble as well as help them focus more on the pedagogical aspects of their new site. Thus, as part of the High Wired project we also wanted to provide a MOO software package specifically designed for educational use. This software package was eventually named the High Wired enCore, or simply *enCore*.

Building on Open Sources

As I have stated above, when we built Lingua MOO, we used the LambdaCore software. This MUD software was designed with a built-in programming language that allowed us to expand and adapt the system to a new set of specifications for academic use to which I will return shortly. The fact that the technology placed this powerful programming tool at our disposal was, and still is, one of its major advantages. The other major advantage that MOO offered was that its code was openly available to anyone who wanted to modify or change it. So while keeping the core functions and compatibility with other MOOs based on LambdaCore, we could easily expand and build in new functionality and remove functions that we deemed unsuited in an academic setting. For these reasons, as well as from a technical point of view, it was clear to us that MOO offered the best and most flexible starting point for building the academic online environment that we wanted.

The enCore MOO software project began in the summer of 1997 and is still ongoing at the time of this writing (2003), with new versions and updates being released as needed. In what follows, I will focus in more detail on the goals and design objectives of this project.

High Wired enCore: An Educational MOO Core Distribution

In 1997 there were basically two MOO software packages that could be used as foundations for building new MOOs. The first one, LambdaCore, was discussed in the previous chapter. The other core package was JHCore, developed by Jay Carlson, Ken Fox and others at a MOO called Jay's House MOO. Both of these packages can be described as very generic and to some extent bare-bones; yet it was clear to us that if we wanted to encourage the adoption and use of MOOs in educational settings, we had to provide a core package that was not only more rich in the kind of features that educators wanted, but that also made it easy to get started for non-technical people.

The primary design objective of the enCore project, as formulated in the summer of 1997, was therefore to create a MOO software package that was:

- Designed specifically for educational use.
- Easy to set up and administer for "non-techie" educators.
- Built-in suite of popular educational MOO tools.
- Used an Open Source model of development to ensure that users have the freedom to modify and adapt the software.
- Available to anyone free of charge.

By this time, LinguaMOO was already a mature, educational MOO system with many new tools and enhancements, so it was evident that it would make a good foundation for a new core distribution. As it turned out, however, instead of extracting the new core from the Lingua database, we ended up using the then latest LambdaCore distribution, which was 02-February-1997. During the summer of 1997 we ported over most of the tools and features that we had added to Lingua MOO, and we also begun soliciting contributions from other MOOs. Back in 1997 the MOO world was still dominated by the traditional text-based interface, but with the rapid rise of the World Wide Web and all the possibilities for multi-media content that it opened up, we felt that it would be very useful to have a nice interface to the WWW in addition to the traditional text-based interface. At the time, one of the most advanced MOO web interfaces was a system called BioGate, originally developed for BioMOO by Gustavo Glusman, Eric Mercer, and others. The BioGate system was much more advanced than the simple browse-only web interface that we had obtained from Collegetown in that it enabled users to browse and interact in the same session. By this time the new Java programming language, with its client side applet, was beginning to take off, and a number of people were experimenting with ways to build telnet applets that could be used in an integrated MOO web interface. When the BioGate system was ported to Diversity University MOO (DU) sometime in 1995-96, Eric Mercer and Alex Stewart, a.k.a. Richelieu, expanded it into a fully integrated MOO web interface, where HTML was used to deliver static MOO content while a Java telnet applet known as Cup-O MUD delivered dynamic real-time content. We wanted our new educational MOO core to have as many state-of-the art tools as possible, so we contacted the BioGate Partners (the group of people behind the design and development of the system) and asked if we could

incorporate their web interface into our MOO core. Their answer was affirmative, but there was one problem. Their license stipulated that while the system was free for educational and non-commercial use, there was a $1000 license fee for commercial users. On the one hand, this might not have been such an obstacle since our target audience would be educational users. On the other hand, pairing our MOO core distribution with the BioGate system would in effect tie us to their non-commercial use license, since if we wanted to provide both systems as an integrated package we couldn't have separate licenses for each part. We hadn't really paid any attention to licensing before the BioGate issue came up, but it demonstrated to us that we needed to decide on a license before we went ahead with the release.

What's in a License?

Ask any software developer and they will tell you that licensing issues are some of the least enjoyable aspects of software engineering. Nevertheless, a license of some sort is a necessary component of any software release to protect the developers' intellectual property rights and liabilities. So we started looking around at what type of licenses other people used. Since we didn't have the money or the desire to hire a lawyer to draw up an elaborate, specialized license for the MOO core distribution, we looked with particular interest at the Free Software Foundation's GNU General Purpose License and also the BSD license. Both of these seemed to offer the legal framework that we sought, i.e., protection from warranty claims resulting from the use of our software. Our initial idea was to ensure that any not-for-profit entity could use our software in any way they wanted, including the right to modify and redistribute their own versions of it. As mentioned previously, the nature of MOO is such that each installation will

take on its own unique characteristics as users and administrators make changes and additions to it. In this regard Free- and Open Source software licenses aptly express the core philosophy of MOO development: collaboration and sharing. As we studied the GNU GPL and the BSD license, it became clear to us that both of them followed closely the tradition established by the original LambdaCore releases where there were no restrictions placed upon usage and redistribution. If someone took our work and made a profit from selling it, so what? While the BSD license was certainly the more open-ended of the two, we felt that it failed to protect one of the key things that we wanted our software to promote—collaboration.

Both licenses protect the users' right to obtain, modify, and redistribute software; but only the GPL also protects the developers themselves. Robert Chassell, formerly of the Free Software Foundation, explains that in the GPL you "have the freedom to do things, but also the freedom from other people taking things away from you. It's the freedom from that ensures that you reward the good guys and not the bad guys" (Chassell "Interview"). By using the GPL, we could prevent users from taking our code and adding their own proprietary elements to it. Any works derived from our software would have to be licensed under the same terms as the original software, which meant that changes and modifications in one enCore MOO could freely be shared with other MOOs. In other words, the GPL protected us from people who might otherwise want to take advantage of our work without being willing to channel their own contributions back to the community. Although we didn't grasp the full significance of it at the time, the choice of the GNU GPL license was a major milestone for the whole project. It provided a legal framework for our work and also a philosophical foundation for the kind of community we wanted to foster.

enCore: The Early Days

The first beta version of enCore was released on the Internet on August 10, 1997. Since the software was provided free of charge, we could not, for obvious reasons, provide the kind of technical support that a commercial company could. However, it was clear that despite our efforts to make the enCore system as intuitive and easy to use as possible, users would inevitably encounter problems and bugs. Among the first things we did following the release, therefore, was to set up an email discussion list with the purpose of creating a virtual "help-desk" where users could help one another.

Like so many other software projects, enCore evolved in a cyclical fashion. As I have just mentioned, we started with an initial design and brainstorming phase where the foundations of the system were laid out, and since we decided to build on the core of Lingua, we had most of the features we wanted already in place. Because of this we had the great advantage of being able to ship the first version very quickly. The second and most important phase, however, was the circular design, implement/re-design, re-implement phase. With the advent of the Internet, software products can be sent to market with literally one keystroke. Because you can reach your users easily, this means that they can also reach you. The advantages of this are substantial. Not only can your product be spread and tested among a wide group of users really fast (public beta testing), you can also get unsolicited comments, feedback, and feature requests at a stage when the software is still in development and at a time when it is still possible to actually incorporate them into the design and implementation of the product. Thus, during the whole enCore project, we've adopted an active beta-release program that allows us to continually fix bugs and add new features while giving our

users frequent and timely updates of the software. The enCore mailing list has been instrumental in providing us with valuable user feedback, and many of the feature requests that have come across the list have subsequently been included in the enCore distribution.

enCore exchange

The Open Source model of development means that anyone has the right to see a program's source code and the freedom to change and modify it as they wish. Since the enCore MOO database is essentially a source code structure with no compiled binaries, giving users access to the source is a natural thing. In fact, the MOO system is set up to allow users to read source code by default. If you wish to prevent them from doing so, you have to disable the read-flag for each of the more than 2400 verbs (MOO programs) in the distribution. We knew that once people started to download enCore and use it to set up their own MOOs, they would want to start making modifications and additions to it. Some of these modifications might be bug fixes that could benefit the whole user community. Additions might include new features that MOO users elsewhere would also want to use. Through postings on the enCore mailing list we encouraged sharing as much as possible; and in order to facilitate this, we created a special section of the enCore web site called the enCore exchange that was intended to be an archive of MOO code that anyone could download and install on their own sites. As a special incentive for sharing, we also let it be known that the best contributions to the enCore exChange would be included in the enCore distribution whenever we released new versions. We had great hopes for the new code archive. This was during the heyday of text-based educational MOOs, and new installations sprang up almost every week. As time went on, however, we

came to realize that it was much harder to get contributions from the outside than we had anticipated. One reason for this was that whatever collaboration there had been in the past increasingly became a victim of what seemed like a beginning rivalry between installations and competition for users. Although there was little or no monetary gain in operating a successful MOO with a large user-base, it did give its administrators and host institution a certain reputation or perhaps "cultural capital" within the academic world. Another, more significant reason for the seemingly lax attitude toward inter-MOO collaboration, however, was the fact that most of the new enCore MOO operators were not programmers or even technical specialists. Most of them were in fact graduate students or faculty members from the humanities and the social sciences. Most of them ran vanilla installations with few or no core changes at all; and the few that had actually made modifications often felt that what they had done was not significant enough to contribute it back to the community. Although we did receive a handful of unsolicited contributions, many of which made their way into the enCore distribution, most of the code in the exchange archive was either written or solicited by us. One example of code that we actively solicited during this time was Amy Bruckman's MacMOOSE utilities, which was a set of MOO verbs and modifications that enabled users to take advantage of the advanced editing features in her MacMOOSE MOO client program (Bruckman "MacMOOSE").

In retrospect, the enCore exChange was an interesting experiment because of what it told us about the differences between trying to run an open source type project in a humanities setting as opposed to doing it in the traditional computer science world. The enCore exChange archive was up for a couple of years before we finally decided to take it off-line in the spring of 1999.

From: "Jan Rune Holmevik" <jan.holmevik@hedb.uib.no>

Date: Mon, 31 May 1999 23:27:03 -0500

To: enCore list <encore@utdallas.edu>

Subject: enCore exChange Closing Its Doors

PUBLIC ANNOUNCEMENT

MAY 31, 1999

It's with regret that we announce today that the enCore exChange MOO code archive will be closing its doors at midnight May 31, 1999. This archive was established in 1997 to promote sharing and collaboration in MOOs. See the original mission statement below.

enCore exChange: The enCore exChange is an archive of free MOO code for sharing among MOO administrators, programmers and others. The site is designed as a clearinghouse for MOO code that up to now has been dispersed across the entire spectrum of MOOs. With enCore exChange we aim to promote a sense of connection, a spirit of collaboration, and a commitment to a pedagogical economy in which educational MOOs advance the creativity and efficiency of teaching in a gift-exchange system of programming.

Over the years we have only received a very small number of contributions to the archive, and it is in light of this that we have decided to cease operations. An initiative such as enCore exChange can only

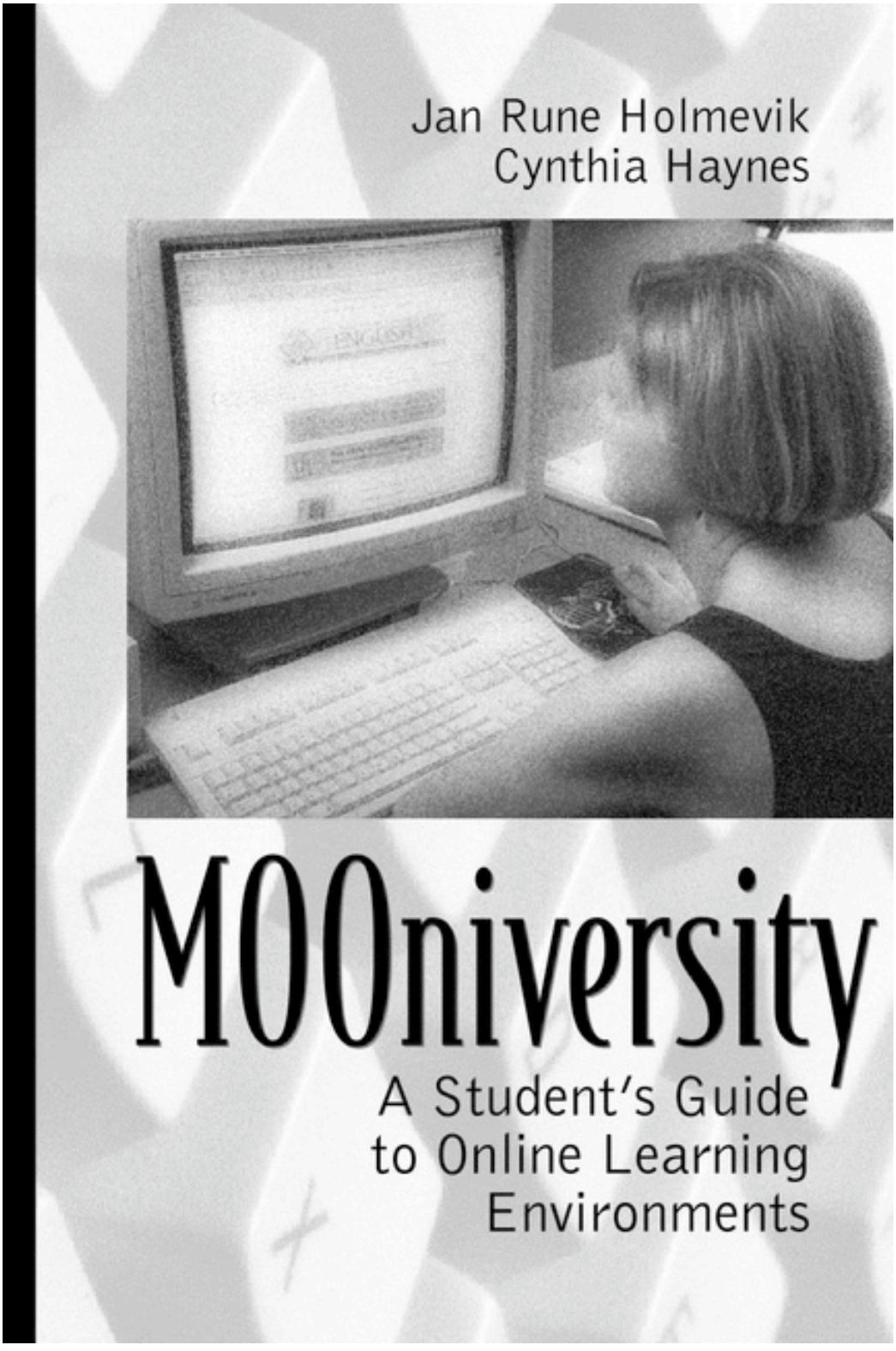
succeed and work to everyone's benefit if people come together to support it. Sadly, this has not been the case.

Regards,

Jan Rune Holmevik and Cynthia Haynes

enCore 2.0: Open Source Stage Two

After the final release of enCore version 1.0 in April of 1998 we turned our attention to the predominant users of the MOO, the students, to see what we could do to make the system easier to use and more appealing to them. Because we had both been using Lingua MOO in our own teaching since 1995, we had a pretty good idea about its shortcomings; to put it simply, the text-based command line of the 1980s and early 90s did not seem particularly appealing or user-friendly to the web-savvy students of the late 90s. It seemed clear to us that something had to be done to "modernize" the MOO interface. The other thing we found was that there were no textbooks that would do for students what *High Wired* had done for MOO administrators and educators, namely, teach them how to use, and make the most out of, the MOO as a learning environment. The textbook problem subsequently led to a book we named *MOOniversity.* It was the first of its kind, and it was designed to give a comprehensive overview of educational MOOs as well as being a combination text, tutorial, and reference book that could guide students through the world of MOO. We wanted to show students how to use the dynamics of real situations to evoke authentic writing in which collaborative and individual learning are enhanced through conversation, research, real-time events, multimedia presentations, and other interactive situations. The book was conceived of as a complement to standard writing texts

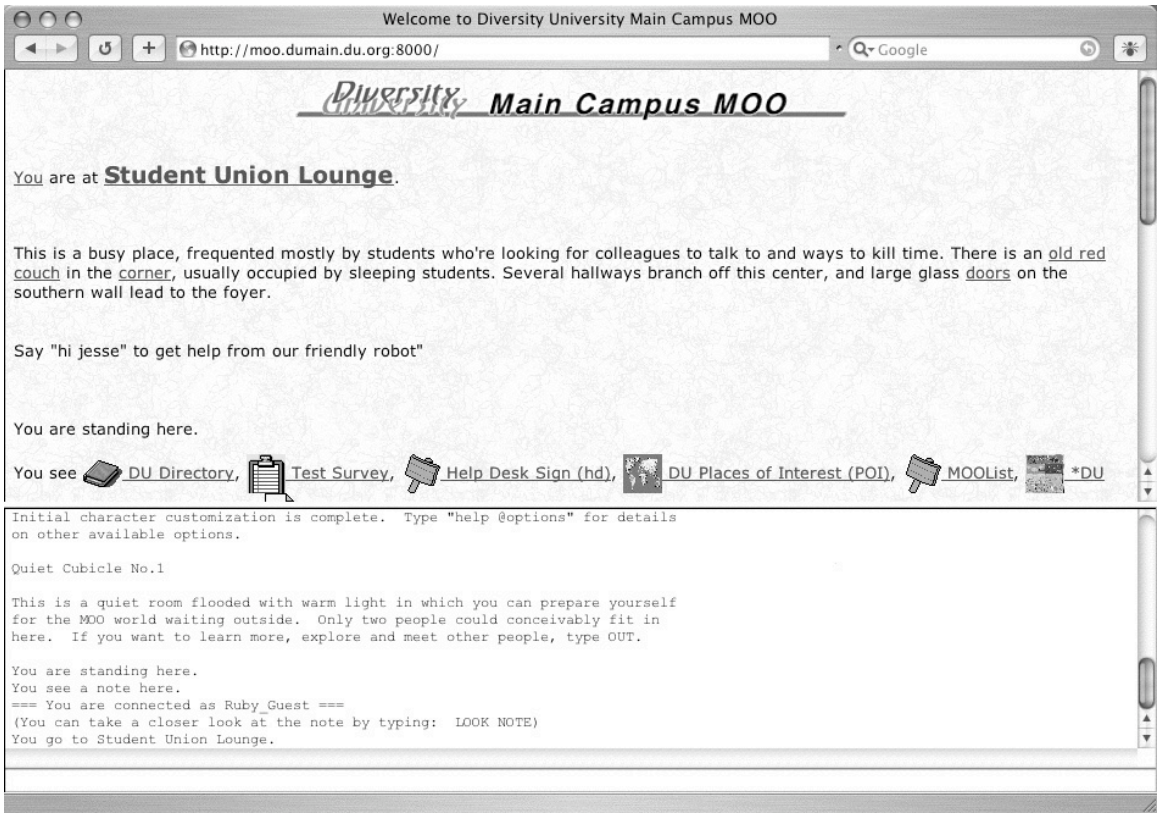


MOOniversity: A Student's Guide to Online Learning Environments. Allyn & Bacon/Longman, 2000.

with clear, useful instructions for writing online, as well as productive assignments and discussion questions to involve students with the technology by writing in real-time with real people. To coincide with the publication of *MOOniversity*, we set about also to modernize the MOO interface, which turned into the graphical user interface project *Xpress* that I'm going to discuss in more detail here. From the beginning, enCore had had a simple web interface that allowed for the incorporation of multi-media content such as images, movies, and sound into the MOO experience. This system was very limited, however, because you could only experience the value-added content while browsing the MOO in non-interactive mode. We wanted to preserve this capability in the new Xpress interface, but we also wanted it to do much more. In outline, our design goals for Xpress were to create a system that would:

- Bring the many hidden power features of the MOO system such as mailing, creating, and editing objects, extending the environment through building, and programming, right to the surface and make them as simple and intuitive to use as possible.
- Preserve and extend the ability to incorporate multi-media content directly into the MOO experience and enhance it by making it accessible directly from inside the real time MOO experience.
- Be platform-independent by being based on World Wide Web technologies such as HTML, Java and Javascript where the Web browser would be the primary software for accessing and using the system.

In enCore version 1.0 we had used a telnet applet by Ken Schweller called "Surf and Turf." This applet allowed users to connect to the MOO and view



Diversity University MOO. BioGate World Wide Web Interface.

graphical content in their web browser; but our experience with it had taught us that students and other novice users often found it confusing and frustrating to have to navigate through many open windows on their screen. In the BioGate system the telnet applet, Cup-O MUD, was integrated right into the main client screen and this made for a much more streamlined and less confusing experience. We contacted Dr. Schweller to ask if we could have permission to design a new applet based on his code, but he politely declined and told us that he was still in the process of working on it and that he would release the source code at a later time.

Once again we found ourselves looking at the BioGate system. As I have already explained, it was clear that we could not incorporate it into enCore due to licensing issues, so instead we began to study the source code to see if we

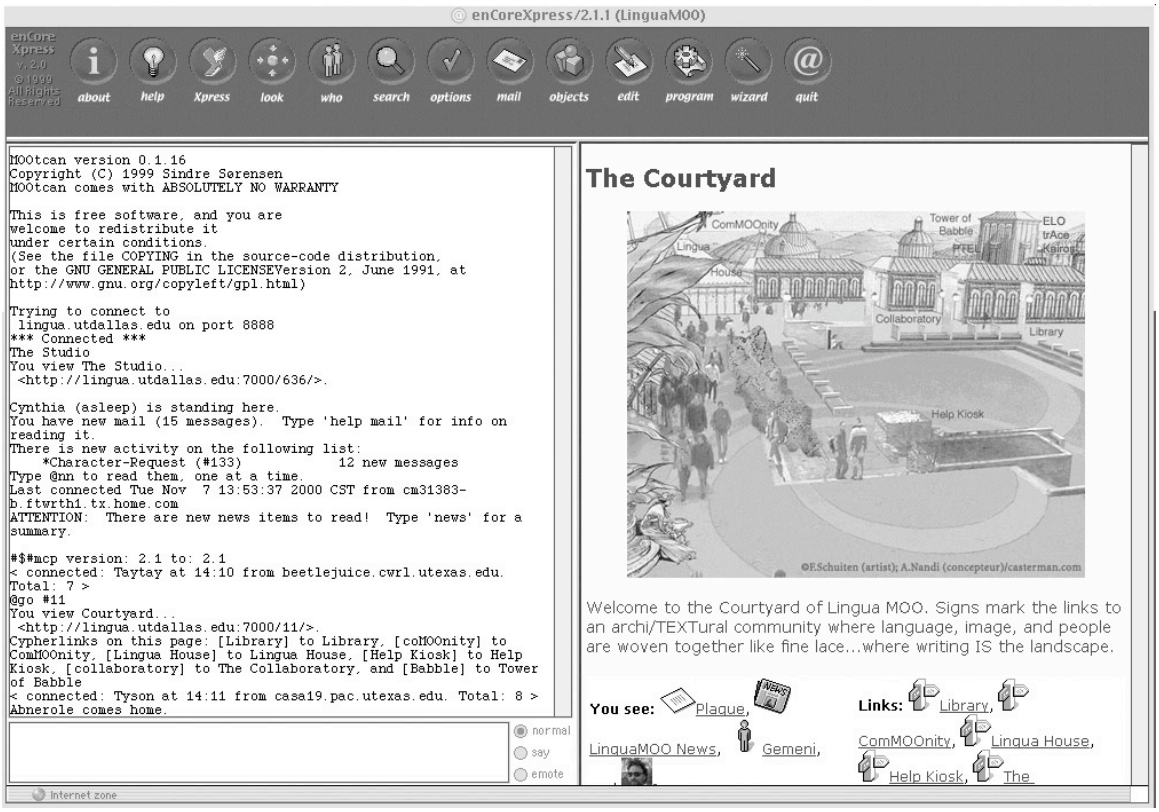
could reverse engineer a new system with the same basic functionality. One of the main strengths of Free- and Open Source software is that you can read and learn from other people's code, and the BioGate code taught us a lot about how to implement an httpd-type web server via MOO code. After a couple of hectic weeks of coding in October of 1998, we had what amounted to a BioGate clone up and running on our development site. It was very basic at that point since the synchronous telnet aspect of it was not yet implemented. We could not go forward with Ken Schweller's Surf and Turf client since we didn't have access to the source, but we had learned about a new MOO telnet applet written by Sindre Sørensen of the University of Bergen, Norway (UiB). The applet was called MOOtcan and seemed to be the perfect fit for the new enCore web interface. Sørensen was at the time affiliated with what was then called the CALLMOO project (later named Lingo.uib), headed by professor Espen Aarseth and Carsten Jopp, also of UiB. Professor Aarseth's project was one of the first to adopt enCore 1.0, and its aim was to use MOO technology in foreign language learning. Specifically, the goal was to create an online learning environment called "Dreistadt" for the German Department at UiB. The project was funded by the Norwegian Research Council and has been one of the most vibrant and creative MOO development projects in the world. After only a brief discussion with Professor Aarseth, it was clear that enCore and the CALLMOO project would both stand to benefit from a closer collaboration; so, while Sørensen continued to improve his MOOtcan applet, we set about to expand the capabilities of the new enCore web interface which later became known as Xpress.

While the BioGate system provided an elegant solution to the integration of MOO and Web, we realized that it had only tapped into a tiny part of what was possible. Our design objectives involved bringing out and simplifying the

creative aspects of MOO, thus making it as easy as possible for users not only to create new rooms and other objects, but also to access the built-in help, mail systems, editors, and more. Another significant benefit of building a web-based client was that it would be platform-independent by running inside a web browser. With the appropriate browser plug-ins, users would be able to broaden the application of MOO to any available web-based multimedia technology. In other words, we could offer a totally new and rich MOO experience while still maintaining compatibility with the traditional text-based system. As educators teaching with MOO in labs, we also knew firsthand about the problems of downloading and installing traditional MOO clients on a large number of machines, not to mention the problems stemming from students trying to download and install said clients on their home or office computers. By using the web browser as the platform for our new MOO client, we could avoid most of these problems because nearly every computer shipped since 1995 had some sort of web browser already installed.

Work on the new Xpress client commenced in earnest in October of 1998, and by January the following year we had the first working prototype ready. During this time we worked closely with the CALLMOO project, and although it was not a formalized collaboration both projects had fairly well defined areas of responsibility. The first beta version of Xpress was a very simple system that only had a few features completed, but it allowed us to test the overall design on a select group of users in Lingua MOO and solicit their feedback. By April of 1999 we had a useable system with most of the features we wanted included, and on April 15th we released the first beta version of enCore 2.0 with Xpress.

In keeping with the Open Source model of development, which says to "release early and often," we shipped several beta versions of the enCore that



enCore Xpress 2.0. The Xpress Graphical User Interface with MOOtcan (left)

spring, fixing bugs and adding new features. The first complete version of the system to include the Xpress interface was released as version 2.0 on June 1, 1999. By this time we had already noticed a considerable increase in traffic on the enCore list due to new subscriptions and also correspondence from new MOO administrators who wanted to try out the new enCore. Although the enCore exChange project had failed to generate the kind sharing and collaboration that we had hoped for, we had not given up on the basic premise. By then the ideas of Free Software and Open Source were everywhere thanks in large part to the success of Linux and the other free software technologies that I have discussed in previous chapters. Also the renewed interest in MOO that the Xpress system was generating encouraged us to try and achieve our goals via a different route, namely, the enCore Open Source Project, which was formally launched in the fall

of 1999. For the purpose of this project we totally redesigned the enCore web site and wrote a new set of instructions on how to contribute, in addition to a "manifesto" which outlined the basic ideas and goals upon which the project was founded (See Appendix A). The site was modeled after other similar Open Source sites with specific sections for information, downloads, and updates. I will return to a discussion of the experiences from the enCore Open Source project at the end of this chapter, but will focus now on the Xpress MOO client.

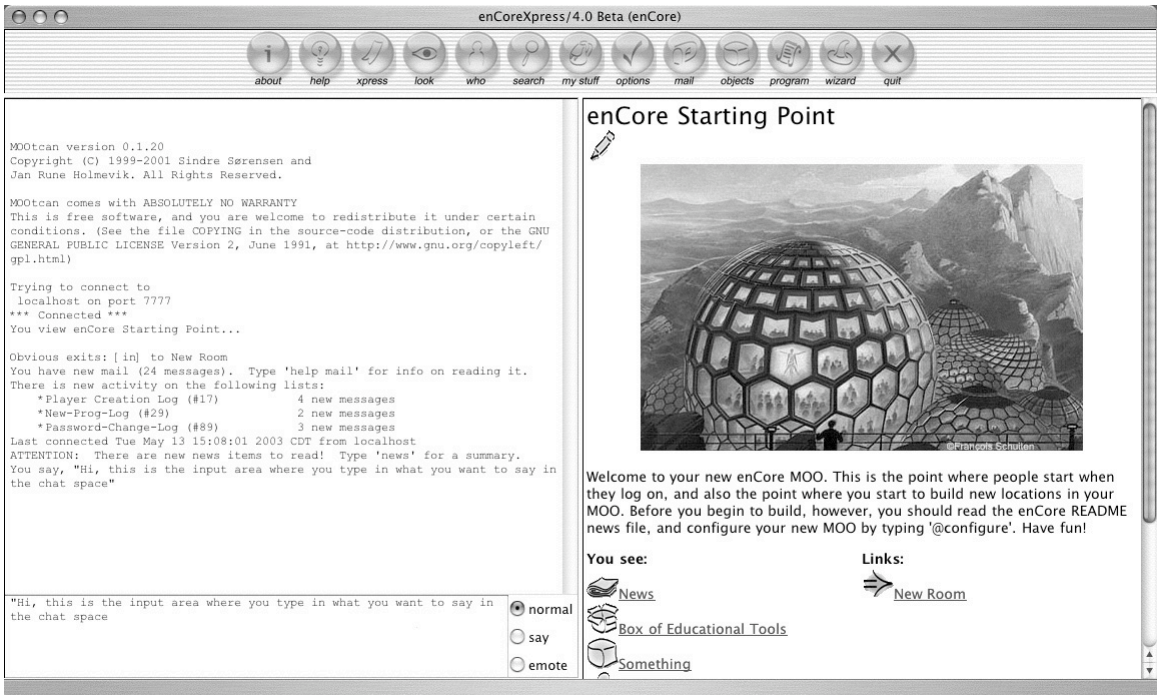
Xpress Explained

In essence, Xpress is a web application that uses a combination of MOO, Java, JavaScript, and HTML (XHTML in enCore 4) to create an integrated MOO client that targets the web browser as its platform. The back-end of the Xpress system is a simple web server implemented inside the MOO using the MOO programming language. Whereas a typical web server such as Apache will fetch and return files from the local file system, the Xpress web server will retrieve information about objects in the MOO database. The web server is supported by a large number of services, such as authentication, error handling, document layout routines and more, also implemented using the MOO language. Java was used to program the telnet applet that handles synchronous actions and events within the MOO. JavaScript is used primarily for window management, while HTML is used for on-the-fly mark-up of information. To understand how Xpress handles a typical transaction between a user and the MOO, consider the following example.

When a user clicks on a link inside the MOO, or types a command to either look at an object or move to a new room, the Xpress web server, internally referred to as $httpd, will receive a request for information over the HTTP 1.0 protocol from the user's browser in the form of a URL such as this one:

http://localhost:7000/62/. Once the request has been received, Xpress will first attempt to determine what exactly the user has asked for. In this case it is object #62 as specified in the URL. If this is a valid object, Xpress will then check to see whether the user has permission to access that object. The authentication procedure takes place in a separate verb (method) and returns a Boolean value, true or false, which is then used to determine further action. If the user is logged in and has permission to view the object, Xpress then calls a special verb on object #62's super class named _html. This verb automatically generates a web page for the object with information stored in the MOO database. In this particular case the system first adds an image, then the description of the room, followed by a list of objects and exits in the room. Appropriate server response headers are then added to the page before it is returned to the user's browser and displayed on his or her screen.

The front-end of the Xpress system is a graphical user interface (GUI) that was designed to deliver rich multi-media content as well as enhance the access to, and usability of, some of the MOO system's more advanced features such as mailing, object creation and editing, programming, system administration, and more. In the default configuration the Java telnet applet (MOOtcan) occupies the lower left area of the screen, also referred to as the chat space, while information about rooms and other objects from the MOO database is rendered using HTML/XHTML in the lower right portion of the screen. This area is referred to as the web space. The chat and web spaces are integrated and aware of each other's state so that if one is updated, the other will be updated to reflect any changes that occur. For example, if a user clicks on an exit link in the web space, their character will be moved to the new location and the web space updated with information about that new location. When this happens a message will



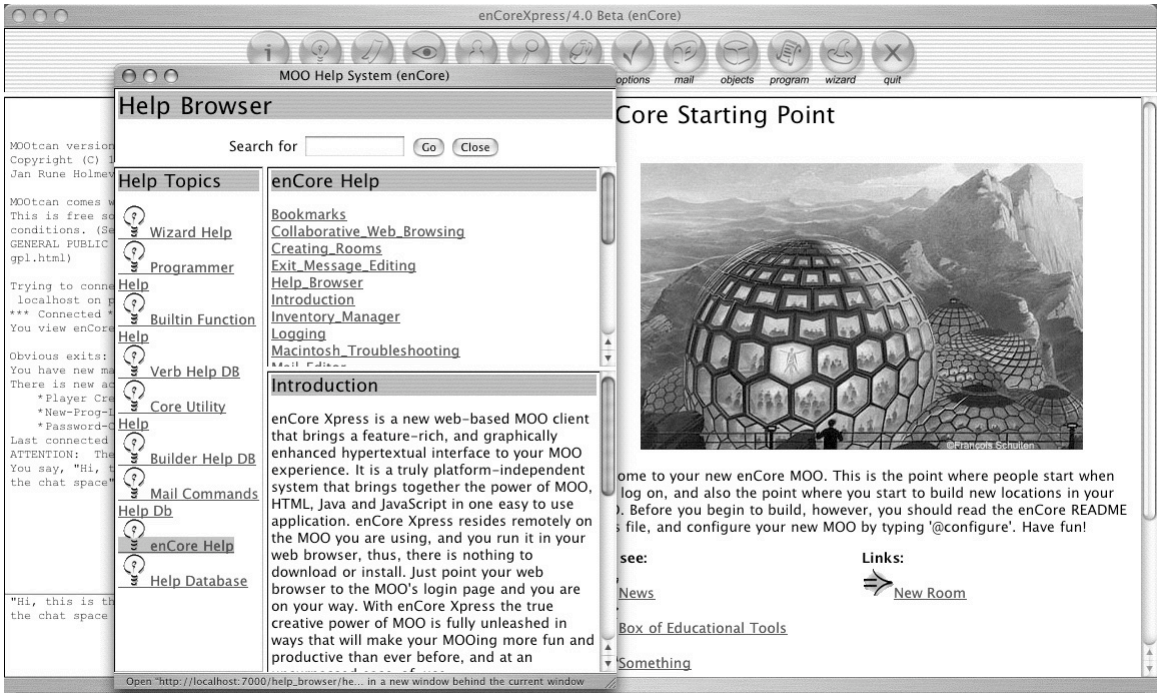
enCore Xpress 3.0. The main client screen.

appear in the chat space telling the user that they have been moved to a new location. They can now engage in conversation with anyone who happens to be located in the room they just entered. Similarly, if a user decides to type an exit command in the chat space in order to move via the traditional command line method, their web space will be automatically updated once they arrive at the new location.

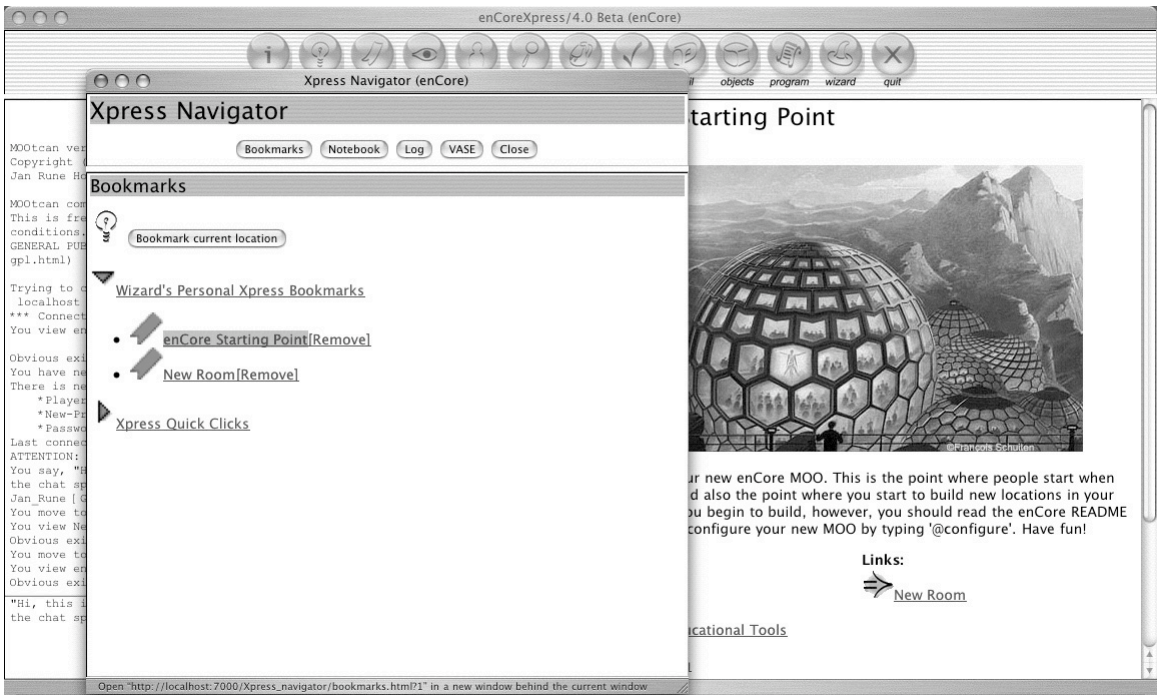
Icons are key elements in any Graphical User Interface, and for the release of enCore 3.0 on April 15th, 2001 we handcrafted over 350 new high definition icons. This was done not only to avoid licensing problems that might arise from the use of downloadable pre-rendered icons, but also to give the system a new and unique look. Since enCore 2.0 users had the ability to change the layout of their client screen from the default vertical orientation to a horizontal orientation akin to the layout used in BioGate, in version 3.0 we introduced themes that also allowed them to change the basic look and feel of the interface.

The Xpress toolbar along the top of the main client screen contains widgets for the main features that we wanted to highlight. In order to reduce complexity and avoid confusion as much as possible, the toolbar is generated automatically based on each user's access privileges in the MOO. For example, a guest character is not allowed to add to or modify the system in any way, so their toolbar will have fewer options than the toolbar that a programmer or a wizard will see. The primary function of the Xpress toolbar is to give users easy access to the more advanced MOO features that I mentioned above. In the traditional text-based command line interface we had found that most of these features were under-utilized because most users simply didn't know, or remember, the commands needed to invoke them. Thus, we observed that the MOO was being used a lot for chatting, class discussions, and teacher/student conferencing, but not so much for creative activities such as building and programming. Even an important resource such as the MOO's built-in help system was hardly being used because users either didn't know about it or didn't know how to access it. A main goal for the Xpress client system, therefore, was to create an interface that embraced these power features and provided easy to use graphical interfaces for each of them.

With the Xpress help browser users can select the help topic they wish to explore, and point and click their way through the comprehensive MOO help database. A list of help topics is displayed to the right in the help browser window. Clicking on one of these help topics will show them a list of key words under that topic. When a user selects the key word they want by clicking on it, the associated text will be displayed in the bottom right area of the help browser window. One can also search for help by typing a key word in the search field in the top area of the help browser window. Another way to learn more about



The Xpress help system.

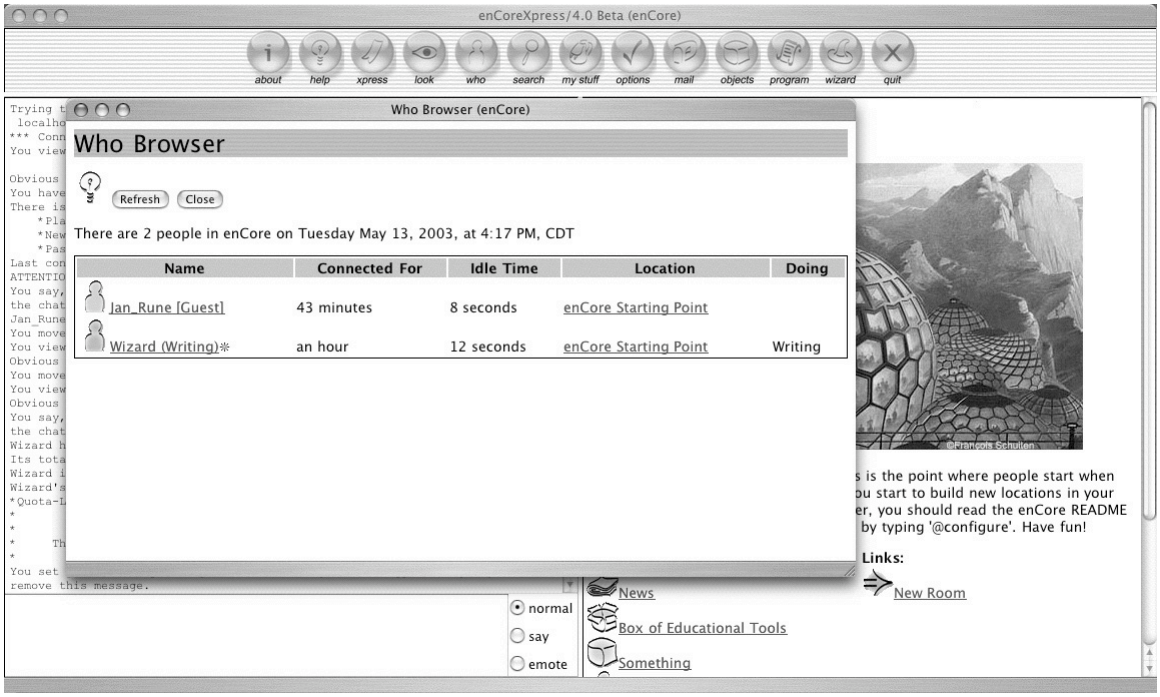


The Xpress Navigator.

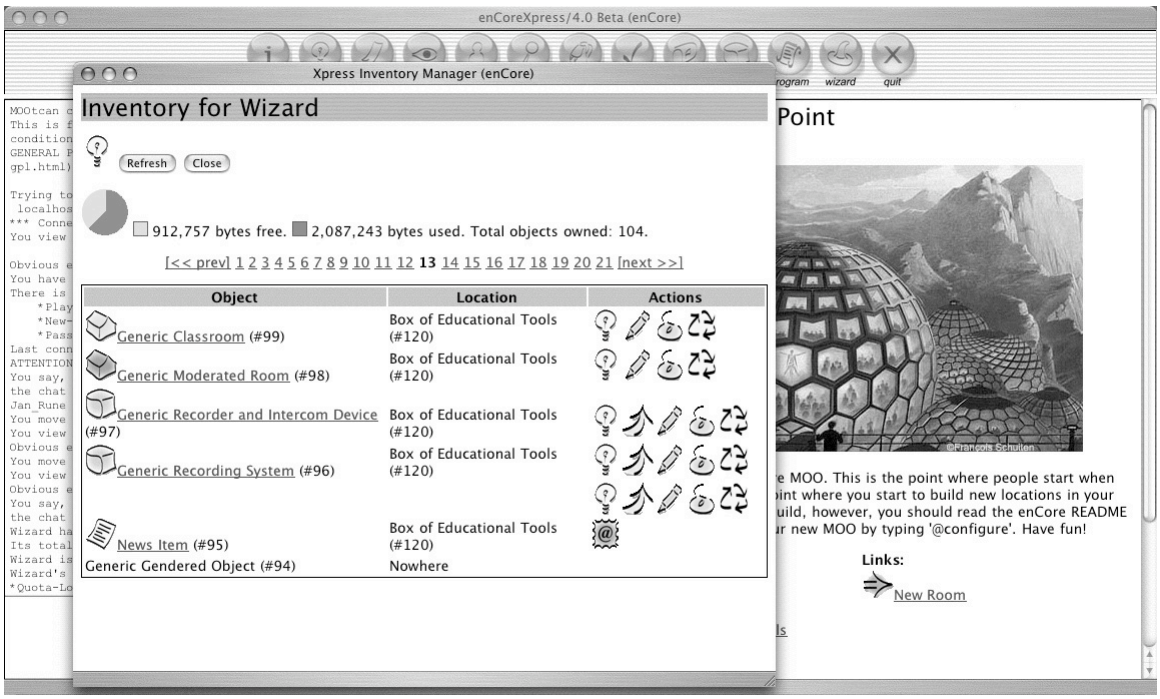
the functions and features available in the MOO is to take advantage of the contextual help system. Many of the objects that users encounter as they wander around the MOO have detailed help texts associated with them. In enCore Xpress, a light bulb icon is used to indicate that further help is available. Clicking on the light bulb icons will bring up help and information relevant to the object.

The Xpress Navigator is a special interface for some of the new features that we added in enCore version 3.0. It gives the user access to a convenient bookmark system that allows them to move quickly between various spaces in the MOO, plus they have the option to add their own personal bookmarks. The navigator also holds a notebook that allows users to take notes from classes, meetings, etc. They may either save the notebook text on the MOO or download it to their own computers via email. A special log feature was added to give users a convenient way to record real-time interaction in the MOO such as meetings, interviews, and so forth. The Virtual Assignment Server Environment (VASE) is a complete online, MOO-based assignment system. With VASE, educators can easily set up assignments for their students to work on online. Students complete and hand in assignments online, making this an ideal distance learning tool within the social learning space of the enCore MOO. VASE was designed and written by Project Achieve of the University of Toronto, Canada.

The Xpress Who Browser shows a list of people currently connected to the MOO. It shows how long users have been connected, how long it has been since they were last active, and where they are in the MOO. One can click on people's names to find out more about them, or join them by clicking on the room they are in. Generic icons are used to denote different player classes, but users can also use their own personalized icon, which will then be used system wide.



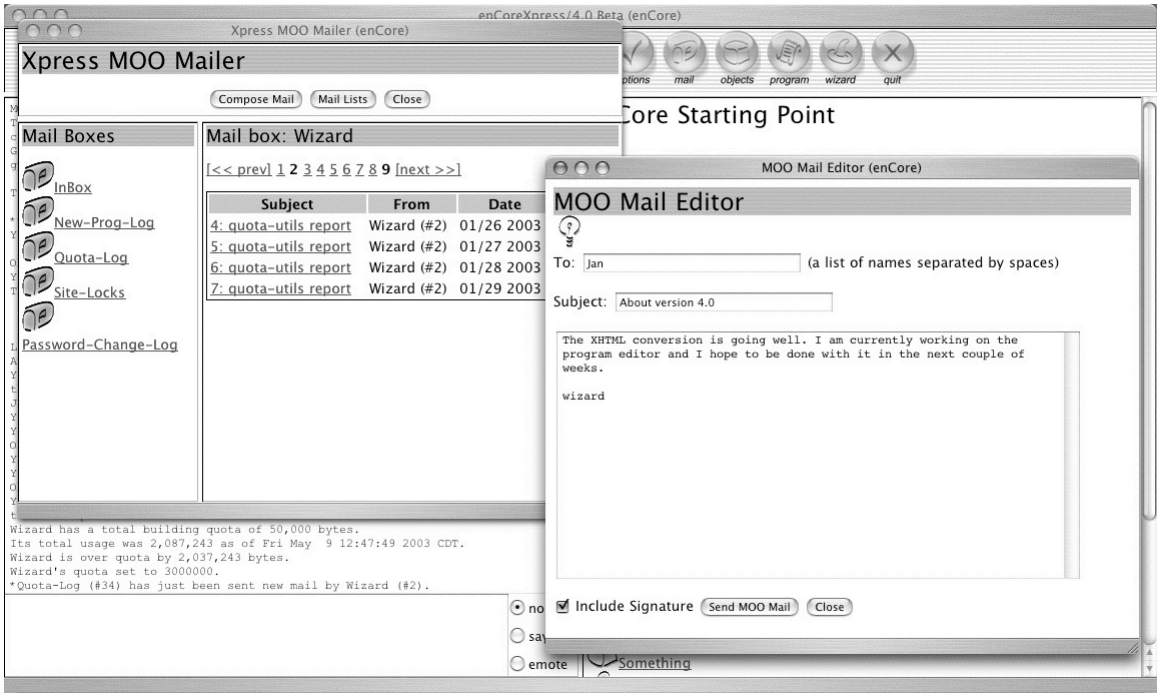
The Xpress Who Browser.



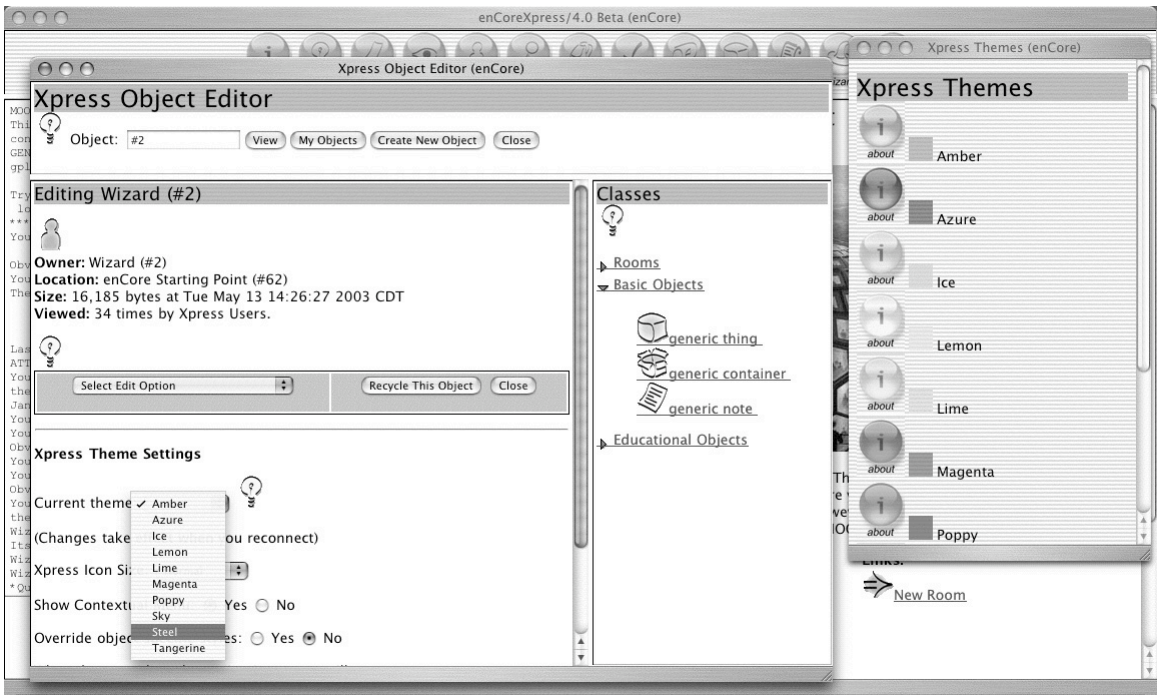
The Xpress Inventory Manager.

The Xpress Inventory Manager gives users an overview of objects that they own. The interface also allows them freedom to manipulate their objects via point and click. For example, they may access help and information about their objects, they can pick up or drop certain objects, edit their objects, lock and unlock them, recycle them, or if the object is a note, they may email it to themselves. The Inventory Manager also shows users at a glance how much building quota they have left. Active MOO builders will sooner or later use up their initial quota, and they must then contact a MOO administrator in order to get more. To avoid screen clutter, if a user owns more than a handful of objects, the list is broken down into several parts. The newest objects will appear first, and one can easily jump to older objects by selecting a new segment from the quick access menu, which appears below the quota information. Certain objects cannot be viewed or manipulated in Xpress. These objects are still listed in the Inventory Manager; however, they have no links or icons associated with them and must be used via the traditional chat space command line interface.

The MOO mail and news group system is a powerful feature that allows users to send each other electronic mail and/or subscribe to MOO news groups. The Xpress MOO mailer interface shown here was designed to make the system more accessible to novice users. A user's inbox, along with any news groups to which he or she may be subscribed, appears in the leftmost area of the MOO mailer, while any messages they have appear on the right. Clicking on a mailbox will bring up the contents of that box in the right hand frame. Users can read any of their messages by clicking on the subject links that appear. To compose a new message a user clicks on the button named "Compose mail," which then brings up the MOO mail editor as shown above. In addition to each user's personal inbox, the MOO may also have a number of news groups that they can subscribe



The Xpress MOO Mailer.

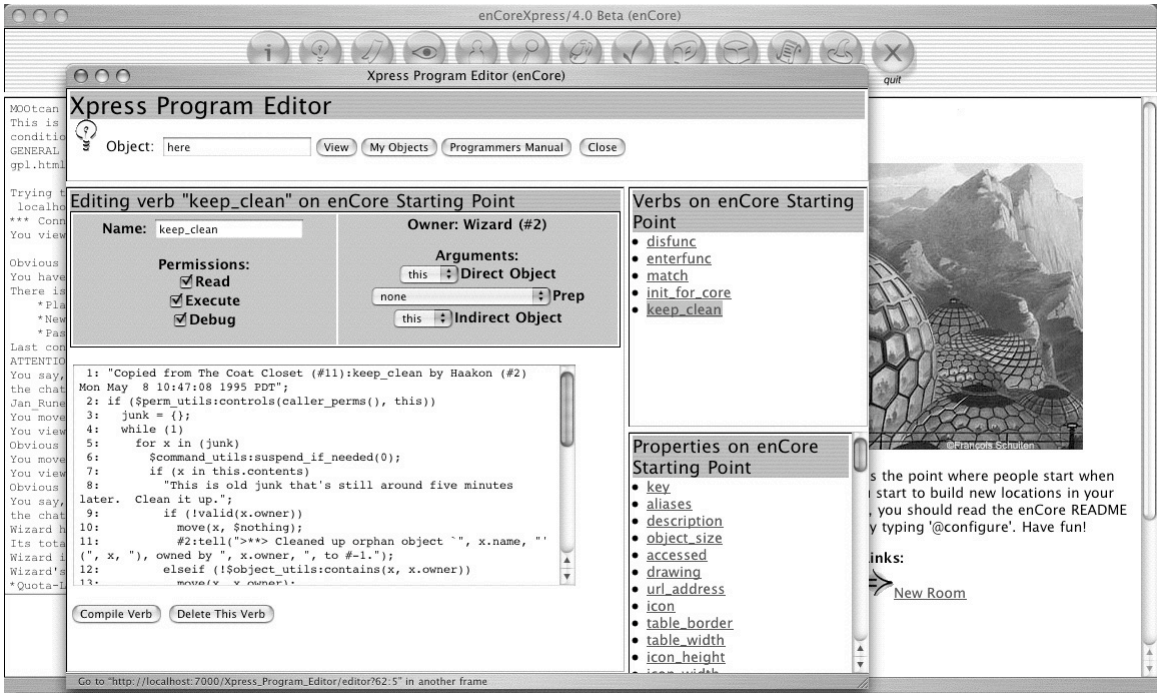


The Xpress Object editor.

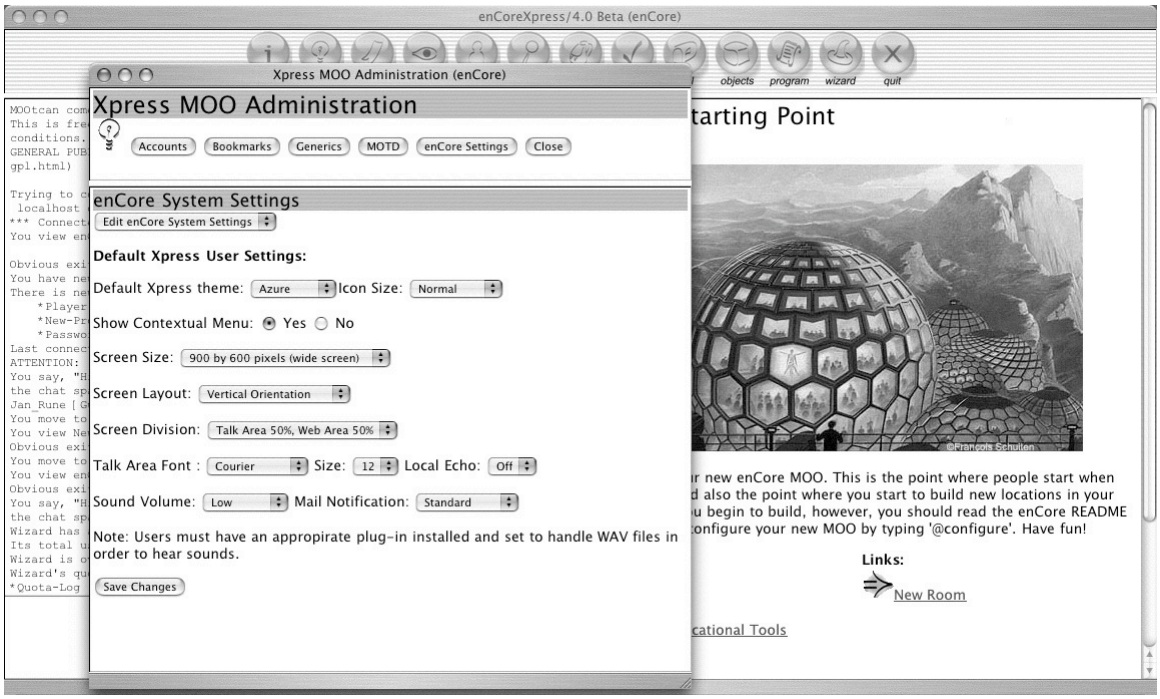
to. Available news groups will show up when users click the Mail List button in the Xpress MOO Mailer. One can subscribe to, and unsubscribe from, mailing lists simply by selecting and deselecting them from the set of available lists shown.

The Xpress Object Editor enables users to create new objects via point and click and also edit some of the most important properties on their objects. The kind of properties that a user can edit with the Xpress Object Editor will vary depending on which class of object they are working with, but here are three of the interface-related properties common to all objects in an enCore MOO.

Multi-media Content: All objects can have external multimedia content associated with them. This might be an image, an audio or video file, a Java or Flash animation, and so forth. One may connect two such resources, audio and visual, to an object. Audio content remains invisible and will play automatically whenever someone looks at the object. Visible content is shown in the Xpress web space. If the size of the display area does not fit the image or embedded resource, one can change the height and width specifications. Leaving one or the other blank results in the browser sizing the image based on the size given. For embedded contents, such as Flash animations or Quicktime movies, both height and width must be specified. Multi-media content requires users to have the appropriate plug-ins installed in their browsers. For this reason, users are encouraged to use only the most common file formats for their multi-media content to ensure that people will see it without having to download and install additional plug-ins. If the external resource is a web page, a link to it will be added to the description of the object.



The Xpress Program Editor.



The Xpress MOO administration module.

Icon: Icons are used to make it easier for people to see what kind of objects they are dealing with. If users don't want to use the default system icons, they can use their own by providing a URL pointing to an external icon-file.

Appearance: Users can control how their objects look like by editing their appearance. They may either choose from pre-defined options for font type, size and color, link colors, background color or image, or they may opt to link to an external cascading style sheet of their own making.

The Xpress Program Editor provides an easy and intuitive interface to programming in the MOO environment. The MOO comes with a built-in programming language that can be used in many creative ways, and the Xpress Program Editor was designed to make it as accessible as possible. The Program Editor lets users look at verbs and properties on objects, edit verb code and property values, as well as manipulate various permissions and flags. To use the Program Editor, users first specify the name or number of the object that they want to work with in the input field labeled Object in the top Program Editor menu. They can either type in the name of an object or an object number. Special MOO variables such as 'me' and 'here' are recognized. Object numbers, however, work system-wide. The Program Editor can also recognize special verb and property references. If a user wants to edit an existing verb on an object, they can type in the object name or number, followed by a colon and then type the verb name (Example: coke:drink). The same syntax is used to reference properties, except a period is used instead of a colon (Example: coke.flavors).

When a user clicks on the View button, the object they have specified will be displayed in the bottom left part of the screen, and any verbs or properties on the object are displayed on the right. To program a new verb, they click the button New Verb, and the Editor opens the new verb ready for editing. If a user's code

has errors in it, the MOO compiler will tell them what kind error was triggered and where it occurred. To create a new property on the same object, they click the button New Property, edit the property value, and click Save Property. To delete existing verbs or properties, users select the item they want to delete from the list of verbs and properties, and click the Delete button in the editor.

Objects, verbs and properties all have certain permission flags that can be set in the Program Editor. For objects, the relevant flags are Read (R) and Fertile (F). If the R-flag is checked, then the object is readable by anyone, but editable only by the owner. If the F-flag is checked, then other people can use the object as a blueprint for creating new objects.

For properties the permission flags are Read (R) and Change (C). Read works in the same way as for objects, while the C-flag determines whether the property can be modified on objects that inherit the property. If, for example, one wants to create a property that should have the same value on all objects that inherit the property, then the C-flag should remain unchecked to make it a constant.

For verbs, the permission flags are Read (R), Execute (X) and Debug (D). If the R-flag is checked, then the verb code is readable by anyone. If the X-flag is checked, then the verb may be called by another verb, and if the D-flag is checked, then the MOO will print a trace back if the verb produces a runtime error.

The Program Editor allows users to inspect all verbs and properties that are readable by them (i.e., all that have the R-flag checked). This means that they can look at readable verbs and properties on other people's objects, but they will not be able to modify them. However, looking at other people's verb code, in particular, is a good way to learn how to program in the MOO; so aspiring programmers are encouraged to explore and learn as much as possible. The

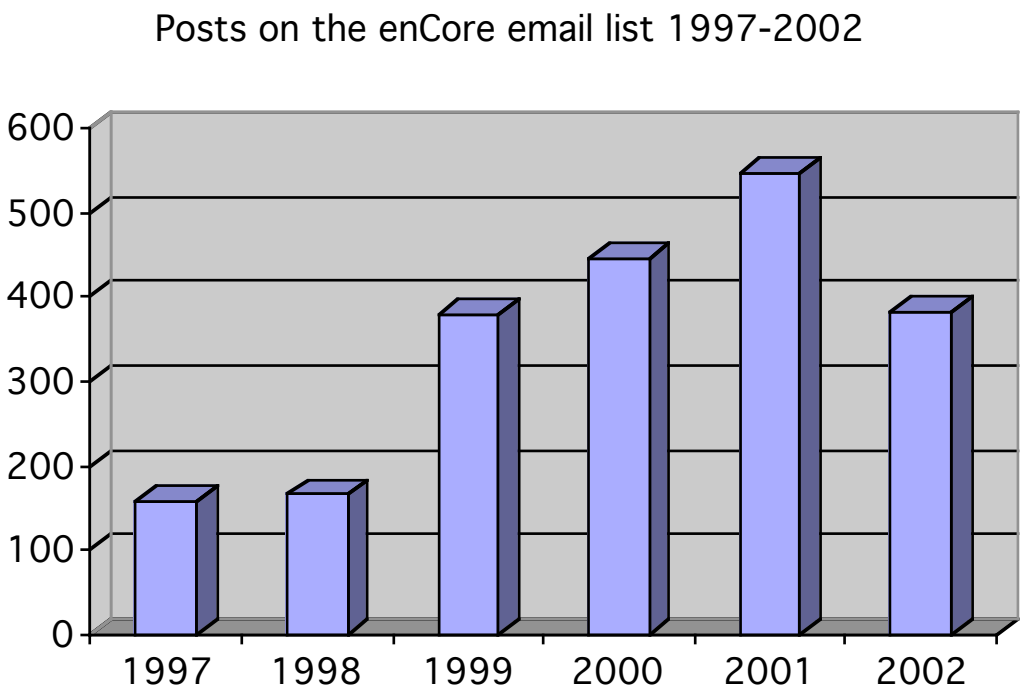
Program Editor comes with Pavel Curtis' LambdaMOO Programming Manual as a reference for programmers.

While Xpress was originally designed to make the MOO easier for users, we have also added GUI to some of the most common system administration tasks. These include things like changing system settings, account creation and management, and bookmark and generics management. Although MOO administrators can now perform many common tasks via Xpress, they still need to rely on the command line for more complex and less frequently used operations.

enCore Xpress: MOO Redux

Due in large part to the Xpress system, MOOs have become more accessible and easier to use. The point-and-click interface has unlocked the many powerful features that make this technology different and unique. For these reasons, since the release of enCore 2.0 with Xpress we have seen a genuine revival in the interest in MOO technology. Based on downloads from our FTP site and general traffic on the enCore mailing-list, we loosely estimate that in the period between 1999 and 2003 there were at least 250 large and small enCore MOOs in operation. Combined these systems may have reached as many as 20.000 users worldwide.

EnCore MOOs have been used for many different purposes over the past few years. From general purpose online learning and conferencing environments for whole institutions, to discipline-specific applications across institutional and national borders, to small purpose specific projects, enCore MOOs continue to serve academic endeavors around the world. Appendix B lists some of the most active and innovative enCore MOOs in operation between 1997 and 2003.



Posts to the enCore administrator's email list 1997-2002. The higher number of posts in 1999-2001 is an indication of increased activity related to the enCore version 2.0 and 3.0 releases. The reduced activity in 2002 is an indication that the system by this time had become more mature and stable.

Conclusions

When we launched the enCore Open Source project in the fall of 1998 we had great hopes that MOO users and administrators elsewhere would want to get involved and contribute to development of the software. In retrospect, looking back at the activities over the past few years, we must admit that the kind of collaboration on system programming that we had originally hoped to foster via the Open Source Project has largely failed to materialize. This is not to say that big contributions have not occurred. Both the MOOtcan telnet applet from CALLMOO/Lingo.uib and the Virtual Assignment Server Environment (VASE)

from Project Achieve at the University of Toronto, Canada represent major contributions to enCore for which we are very grateful.

What we have seen instead, however, has been a fairly extensive collaboration on aspects of what we may call system maintenance. Since we adopted an active beta release program right from the start, many of our users have been able to help find and fix bugs and other problems along the way. The enCore email list has been instrumental in this regard. What has typically happened following a release is that users elsewhere start to discover problems that we have not been able to catch during our pre-release in-house test runs. Most of them will simply report the problems on the list so that we can fix them, but quite a few have also been able to provide us with patches to actually fix the problems they have found. In quite a few instances we have also received new and improved versions of code from other MOOs long after the original release date. In both cases we have done our best to incorporate these contributions into the enCore distribution as soon as possible.

Another aspect in which the user community has played an important role has been as a supplier of ideas for how to improve existing features. There have been a handful of feature requests over the years, but not to the extent that we might have expected. But occasionally there are very useful new features and resource archives initiated by enCore administrators. The enCore Documentation project co-coordinated by Lennie Irvin of San Antonio College and Erin Karper of Purdue University is another example of concrete, non-technical contributions from the community. This is an independent project with which we were not involved, yet because Irvin and Karper alerted the enCore list members to this important resource, the enCore project as a whole has benefited from it.

There may be several reasons why the enCore Open Source Project has failed to deliver on our original goals and instead has delivered in other significant ways. One of the principal reasons, I believe, has been that our main user groups have come from the humanities and the social sciences rather than computer science. Most of our users did not become involved with enCore in order to contribute to an Open Source project. Rather, they got involved because they wanted to utilize the services that the enCore system provided. Consequently, most of the enCore administrators over the years have not possessed the programming skills and knowledge to actually contribute to the technical development. Secondly, as Eric Raymond and others have pointed out, only a handful of Open Source projects gain enough momentum and fame to actually attract large number of programmers. Linux and some of the other successful Open Source technologies that I have discussed in previous chapters are really the exceptions rather than the rule. MOO and its programming language are fairly obscure when compared to languages such as C and Java, which are the typical tools in today's open source community. Thus, programmers interested in participating in open source development most likely become involved in more highly profiled projects using tools with which they are more familiar.

A third reason why there haven't been as many contributions to enCore as we had hoped has to do with the nature of MOO itself. When a new MOO is first established, it is basically a copy of the core distribution. As its administrators start to modify and customize it, however, it will start to take on its own characteristics. Unless the programmers take care and plan for their local changes to be incorporated back into the main enCore distribution, as well as document their modifications properly, it is often difficult to extract code from a production MOO environment. Sometimes the changes that are made are so substantial and

localized that it is impractical to move them back into the main distribution. Lingo.uib's MOO, Dreistadt, is an example of this. Much of the Dreistadt core was translated into German in order to use the system for its intended purpose. Consequently, much of that work could not easily have been merged back into enCore. Another related problem is that many of the programmers that have worked on MOOs over the years have been students. When they graduate and leave, the new staff have found it difficult to determine exactly what has been done to the MOO unless proper documentation procedures were in place and followed from the beginning.

Finally, the role of funding, or rather lack thereof, in volunteer-based projects such as this often affects the level of participation. Throughout both the Lingua MOO and enCore Open Source projects, we have built on, and benefited from, the Open Source model of development; but we have not benefited directly from grants or institutional funding. All the development work itself has been conducted on a voluntary basis, and this means that as the project proceeds you are more or less at the mercy of whoever decides to sign up. If no one is paid for their time, you cannot delegate work in the same way as if your collaborators were employed. The most you can do is to announce what needs to be done and then hope somebody will come along and do it. If nobody volunteers, you either have to do the work yourself, or not do it at all. This is a problem for most projects that rely on volunteers. As I discussed in Chapter Three, even a large and well-known project such as GNU had to hire programmers in order to get certain tasks done. Fortunately, we have benefited indirectly from other funded projects working with enCore technology. Lingo.uib's MOOtcan applet is a good example of this. Lingo.uib worked within the framework of a well-funded project

with wide institutional support, and thus they were able to hire programmers to do the various jobs they needed done.

The enCore project has clearly demonstrated that Open Source methodologies afford some truly remarkable opportunities for learning. By building on open sources and existing technology, we were able to learn from those who went before us. We did not have to "re-invent the wheel" in order to create a more suitable program for our needs. Complete access to the MOO source code gave us the opportunity to study how things were done so we could improve and change what we wanted, and also add new features as we went along. MOO technology is truly *free* in both senses of that word. First, it costs nothing to acquire and very little to operate; thus, it is ideally suited to the needs of often cash-strapped educational institutions. Secondly, and even more important however, is the fact that users are free to change and modify it to suit their own needs and specifications.

Another important lesson has been the significance of actual user experiences and input on the design and implementation process. With most software products, users typically have no influence over the design, which usually results in a situation where they just have to make the best use they can of the software as handed to them. In the Open Source model that we used for enCore, not only has the design of the system been informed by our own teacher/user experiences, we have also actively solicited comments and feedback from other users in an attempt to make our software as useful and empowering as possible.

More than anything, the enCore project has shown us that we do not have to settle for commercial products that either force us to work in certain ways or that take away our ability and freedom to be creative and bend the technology to our own needs. Through the Open Source model of development we, as educators in

the humanities, can forge our own technologies and make them work the way we want.

7

Conclusions

en·core **:***a demand for repetition or reappearance made by an audience; also* **:***a reappearance or additional performance in response to such a demand—Merriam-Webster OnLine*

The word encore means repetition or reappearance in response to a demand. This dissertation has concerned itself with two such encores: one, the popularization of hacker methodologies in the 1980s and 90s; the other, the re-creation of educational MOO technology in the shape of the enCore Xpress system. While Open Source may be seen as an encore to Free Software and the hacker practices of the 1960s, 70s, and 80s, and each Open Source program an encore to the one on which it was based or inspired, enCore Xpress can be viewed as an encore to educational MOO technology. Although neither of these encores were exact repetitions of previous efforts, they were certainly reappearances in response to a demand from the groups that employed them.

My dissertation began with one basic question in regard to ICT; how can we as scholars and educators in the humanities become more actively involved in the conceptualization and creation of our own technological future? If we believe that we possess valuable knowledge and experience in regard to teaching and

research in our respective fields, then we ought to shift our academic mission to include infusing the technologies we use with our knowledge and experience. Through this project I have turned my initial question into three distinct challenges and then *enacted* these challenges in an effort to demonstrate one way in which we can enhance technology with our own unique knowledge and experience. The three challenges that I outlined in Chapter One can be summed up in the keywords *use*, *study*, and *enact*. These terms form the basis of my final remarks since they have shaped the experiences and lessons of both the enCore project itself as well as this dissertation.

Practice What You Preach

Change stems from experience, and experience stems from use. My involvement with Lingua MOO from 1995 onward has taught me a great deal about teaching with ICT. One lesson that I have learned is that we cannot just introduce new technology in our classroom and expect to continue our teaching practices as before. If we do that, we will miss many of the new and unique opportunities that modern information and communication technologies can provide. On the other hand, if we try too hard to bend our teaching practices to fit the mold of a certain technology—if we let the limitations of the technology itself govern our learning strategies—then we end up with a contrived and quite possibly counter-productive learning environment. Sometimes we do come across learning technologies that are perfect for our purposes; more often than not, however, we make do with what is available.

Many of the technologies that are currently being used in learning institutions around the world restrict involvement from third parties. They are proprietary and closed to everyone except those who create and maintain them. As I see it,

we have only two options if we want to influence development of such proprietary technologies: 1) we can contact the developers and give them feedback on changes we want them to make; or 2) try to go into partnerships with the developers so we can influence the technologies more directly from within. Neither of these alternatives is optimal for our purposes. Trying to initiate change through bug reports and feature requests is a gamble that leaves us at the mercy of those who produce the technology. Trying to insert ourselves into corporate structures such as a software engineering company is a lot easier said than done. In reality, therefore, working with proprietary technologies is not very helpful if our ambition is to institute change. For this reason it is my belief that we ought to rid ourselves as much as possible of proprietary solutions in favor of the many open source alternatives that now exist. These open technologies invite us to play with them and learn from them. They empower us on many levels, the most important of which is that they allow us to change them.

The Promise of Open Source

Software is a strange thing. It is immaterial and abstract—it is the conceptualization of how we solve problems in a digital world. Computers and networks are material things, but they are nothing more than empty vessels until we fill them with software. Understanding software is key to understanding the information society in which we live. The purpose for studying software in the context of this dissertation has been to learn about the methodologies and practices that the hackers have developed in the creation of software.

In Chapter Two I discussed the emergence of software and how it was made possible by the invention of the universal machine—the modern computer. Until

the computer, what we now think of as software, namely what makes a machine do what it does, was encoded directly into the hardware itself. The universal machine created a separation between the machine and its application space, and this effectively gave birth to what we might call the modern notion of software. Unlike the first digital computers of the 1940s and 50s, which were enormously expensive, the software created for them was, by contrast, quite inexpensive. Once the cost of actually writing it had been absorbed, duplicating the software cost next to nothing since it was digital and electronic by nature. During this time few considered software development to be a commercially sound enterprise by itself—the money was in hardware and software was primarily a means to sell more computers. Much of the research and development of software during this time was academic in nature, and academics have always shared knowledge among themselves. The Multics operating system project, which I discussed in Chapter Three, is a good case in point. The knowledge, including the technical concepts, that this project generated was later used in the creation of Unix, which in turn gave the impetus to BSD, GNU, and ultimately, one might argue, Linux. Thus, when the first hackers came along in the 1960s they entered a world in which software, on the one hand, was not generally considered a commodity, and on the other, something for which there was already precedence for sharing.

In Chapter Two I made the case that the invention of the personal computer in the 1970s changed the value of software and, by extension, the way that it was developed. New killer apps, such as computer games, VisiCalc, Aldus PageMaker and more, coupled with cheaper, mass-produced machines led to the formation of a new commercial PC software industry where source code became a highly guarded business secret. Apple Computer, one of the pioneers in the PC industry, built the whole company on the established practices of using software

to sell hardware. Microsoft, on the other hand, never entered the hardware market at all—they focused all their energies and creative talent on software alone. Today Apple Computer occupies less than 2.5% of the worldwide market for personal computers. Microsoft controls almost all the rest. There are, of course, a complex set of reasons why the two companies ultimately ended up on such radically different paths, and I am not at all ascribing it just to their different strategies in regard to software. Still, I do believe that the history of Apple and Microsoft says a lot about the direction in the value and importance of software.

While a large scale commercialization of software was taking place during the 1970s and 80s, hackers continued to refine and extend the collaborative code-sharing practices that had been established in academic software circles. Much of this development took place in connection with the emerging computer networks such as the ARPANET and therefore out of sight of the media and the general public. In Chapter Two I traced the development of a handful of the important communication technologies that helped make the Internet such a success. In Chapter Three, I discussed the development of BSD, the operating system that became such an important component of the Internet's underlying infrastructure itself. As I have pointed out, many of the programmers who were involved in these developments can clearly be labeled as hackers, others employed the collaborative hacker methodologies.

Richard Stallman, at one point dubbed "the last of the true hackers" (Levy 415), played a larger role than most in the history of hackers. When in 1983-84 he set about to create the GNU project and later the Free Software Foundation, he began a process of self-awareness among hackers. He gave them a cause to rally around—the creation of a new operating system free of proprietary restrictions—and he gave them an ideological framework—Free Software—to

which they could relate their identity as hackers. As mentioned in Chapter Four, Eric Raymond points out that Stallman's project was a moral crusade. He wanted to strike back at the commercialization of software, the disempowerment of users, and the restrictions on freedom that went along with it. From a historical perspective we might say that Stallman began molding the individualistic hackers into a self-conscious movement.

When Linus Torvalds came along and created Linux in the 1990s, the emerging hacker movement received a major boost both in numbers and in reputation. The remarkable media attention that the GNU/Linux operating system received in the latter part of the decade made it a perfect showcase for the hackers and their collaborative software development methodologies, and many within the movement were eager to spread their gospel to business and industry, now completely dominated by proprietary practices and solutions. Eric Raymond said, "We knew we had a better way to do things in our software designs and operating systems and the way that we shared work with each other. But we couldn't get anybody to listen" (Raymond "Interview"). One reason, they thought, why nobody in industry would listen had to do with Richard Stallman's ideology of Free Software. In response, they created the new notion of Open Source as a more pragmatic model that incorporated the basic methods and practices of Free Software, but was devoid of its additional ideological baggage. The creation of Open Source in 1998 was also a response to a growing diversification within the movement. While many continued to join in order to further the ideology of Free Software, others simply wanted to become involved with the hacker methods and practices at a purely practical level. Today, the hackers themselves talk about two separate movements, the Free Software movement and the Open Source movement. I believe that these are not

necessarily different movements, but rather political wings within the overall hacker movement. In my dissertation I have used Open Source rather than Free Software as a label for the collaborative model of software development that I have employed.

Unlike proprietary systems, which we are typically not allowed to change, Open Source does not force us to reinvent the wheel in order to create better systems. In fact, hacker practices suggest the exact opposite. Not only should we study and emulate the way hackers learn what they learn from each other, we should allow their passion for learning to encourage us to learn about systems by studying their source code and the structures into which they are built. Once we have acquired an understanding of what a system does and how it works, it further encourages us to play with it and modify it bit by bit until we have changed it the way we want. In the *Jargon File*, Eric S. Raymond defines hacking as "an appropriate application of ingenuity." Open Source technologies invite us to become hackers by applying our own ingenuity toward change. This is empowerment at the highest level. We are invited to learn from those who came before us, and to add our own ingenuity and solutions into the mix in a great collaborative and evolutionary process.

Open Source also represents a democratization of technological development. On the one hand it levels the playing field by allowing us, as scholars and educators in the humanities, to become involved in the creation of technology at levels that scale with our own technological expertise. We do not have to hold a Ph.D. in computer science to contribute productively. Sometimes we may accomplish our goals just by tinkering with and modifying small parts of a system, other times we may have to make more substantial changes. In any event, the freedom to change a system hinges only on our own technical

capabilities and our motivation to learn the necessary programming skills. On the other hand, the democratization also speaks to the use of technology and the important freedoms granted to us in this regard. While commercial shrink-wrap licenses and other proprietary licensing schemes grant creators of technology power over users by placing restrictions on usage and by keeping the source code secret, licenses that conform to the Open Source Definition, such as the GNU General Purpose License and the BSD license, do exactly the opposite.

There is no doubt in my mind that Open Source holds a great promise that we have only just begun to tap into. Having said that, I must rush to add that we should not forget that it is easy to get carried away in the euphoria that has accompanied the present-day Open Source encores to the earlier hacker performances. Contrary to what some of the most ardent Open Source evangelists like to claim, and as I have pointed out in Chapter Four, even in the computer science and engineering fields Open Source has not proved to be a panacea for everything that ails current dominant practices. Thus, it would be imprudent to rush to the conclusion that Open Source is going to be the one and only answer to the challenges that I have outlined for the humanities. Before we can begin to draw any conclusions in this regard we must first conduct experiments where we apply Open Source methodologies to actual problems and challenges in the humanities. This is precisely what I have done with the enCore Open Source project.

Lessons From the enCore Open Source Project

In Chapter Four I discussed Eric Raymond's essay, "The Cathedral and the Bazaar," and it might be helpful at this point to relate our experiences from the enCore Open Source project to some of his key observations from that article.

When we first started our work back in 1995, Raymond's article was still two years off and the Bazaar model as such was therefore unknown to us. In hindsight, however, it is evident that we employed many of the same methods and strategies that he describes as key elements to a successful Open Source project. For example, Raymond asserts that "every good work of software starts by scratching a developer's personal itch," and that, "to solve an interesting problem, start by finding a problem that is interesting to you." As I explained in Chapter Six, this was exactly how the enCore Open Source project started. Between 1995 and 1997, Cynthia Haynes and I identified a number of issues with MOO technology and its applicability to teaching and learning, and based on these observations we formulated several enhancements and new features that we wanted to implement. Although never explicitly cast as Open Source, MOO technology had always been open and free for anyone to use and modify. Again, we found ourselves doing precisely what Raymond describes as the typical hacker's approach to software engineering: "Good programmers know what to write. Great ones know what to rewrite (and reuse)." As an experiment with Open Source methods in a humanities environment and also in an attempt to garner more interest and participation in the enCore development efforts, the Open Source project was then created in the fall of 1998.

Looking at the formidable success of Linux, and the general hoopla surrounding Open Source at the time, our initial hopes were somewhat similar to those of Netscape—that going Open Source would automatically lead to an increased number of outside volunteers. As I discussed in Chapter Six, these hopes largely failed to materialize, and in retrospect we might say that they were overly optimistic. What we, and many other small Open Source projects, have learned is that contrary to popular belief, most such projects do not consist of

hundreds or thousands of programmers. Most of them are, in fact, made up of relatively small numbers of core developers. Many Open Source projects may seem huge on the surface due to their success in the market place. However, the actual numbers of core programmers are not nearly as big as we might think.

A further explanation as to why the enCore project failed to attract more outside programmers, I think, must be attributed to MOO technology itself being rather obscure as compared to better-known systems and programming languages. Given the choice between joining a project that used C, Java, PHP or some other well-known development tools, and ours, which used MOO primarily, most prospective programmers would probably look elsewhere.

Finally, I believe that we failed to attract much contribution of code because most of the people who used enCore to start new MOOs were primarily interested in it from a pedagogical standpoint. Very few of them had any programming experience, and thus they may have felt at a disadvantage when it came to contributing technically. Part of the problem with participation that I have just mentioned stems, of course, from the not too surprising lack of programming expertise in the humanities. This, however, is one issue that Humanistic Informatics as an academic field is perfectly situated to help resolve. Had we conducted the enCore experiment five or ten years hence, the results in terms of programmer participation may well have been more encouraging.

The fact that the enCore system did become a reality and has, over the years, attracted more users than we have been able to track, speaks to a different level of participation that, in hindsight, is an equally important lesson. In the "Cathedral and the Bazaar," Raymond advised that "if you treat your beta-testers as if they're your most valuable resource, they will respond by becoming your most valuable resource." Using an Open Source model of development enabled

us to incorporate our users directly into a rapid and efficient maintenance scheme. From the project's beginning we had an email discussion list that served as a direct link between our users and us. The list was intended as a virtual help desk, but as the project progressed it also became an important channel for bug reports and feature requests. This is something that proved immensely beneficial both from a user and a developer standpoint. Since we had a large pool of active and interested users, following Raymond's observation of releasing early and often was the natural thing to do. From a user perspective it meant that they became actively involved in the evolution of the system and were able to influence its development in very concrete ways. From a developer point of view it meant that we were able to find and solve bugs more easily and release new updates very quickly. It is therefore correct to say that the enCore project has been a collaborative exercise although in a different manner than we had originally anticipated.

The results of this collective effort have unfolded somewhat like what one would imagine goes on before and during an expedition—planning, training, testing, mapping, designing, equipping, and trekking. But no expedition succeeds without the participation of a team of explorers, each skilled in different ways, each motivated as much by the passion for the trek as by the remoteness of the destination. Thus the enCore Portfolio (Appendix B) reads like the map of a new world, with the names of members of this expedition marking the new routes to new regions, writing the journals and logging everything from the pinnacle to the mundane. So while I could not have anticipated the outgrowth of our collective expedition to be so vast, I remain continually gratified that enCore has pushed the edges of this map with each launch of a new enCore MOO, each research publication, each conference lecture, and each student encountering an

enCore in the making. It is therefore *also* correct to say that beyond those users participating on the email discussion list, other users of enCore MOOs, such as teachers, students, deans, department chairs, professional association staff, and writers of all kinds of writing contributed equally to the wellspring that became *the* open pool of *this* Open Source.

The lessons do not end here, however. What inspired me in the hacker movement inspired not only the creation of enCore, it has also inspired a 'hacker pedagogy' modeled from their passion for learning and sharing the knowledge gained. This new direction of humanities research suggests, as Haynes' forthcoming book *Beta Rhetoric* attests, that the intersection of humanities and technology is no longer merely the point where two linear and vertical crossroads meet. Hacker pedagogy is omni-directional—going in all directions at once. While Humanistic Informatics emphasizes what should be learned, hacker pedagogy emphasizes how it should be taught. Lessons learned become lessons in how to teach—how to challenge and change.

A Final Challenge

I would like to end here with one final challenge. While the programming and software development issues that I have discussed on the preceding pages may have been foreign territory to most scholars and educators in the humanities, the study of Open Source can also teach us about activities that are much closer to home, namely writing and publication, which is every humanities scholar's trade.

Hackers not only popularized the Open Source model of software development, they have also helped inspire and pioneer a new collaborative authoring and publishing scheme for traditional texts such as this. The

counterpart to Open Source is the Open Content, and among hackers, sharing documentation of all sorts has been an established practice for as long as sharing of source code itself has existed. In 1998, David A. Wiley, then a graduate student at Brigham Young University in Provo, Utah, observed some of the amazing results that had come from collaborative Open Source practices in the software world. He wanted to see if similar results could be achieved in traditional academic research and publishing. To this end he formed a project named OpenContent and created a new Open Source-type license specifically designed to grant readers of traditional texts the same freedoms as users of Open Source software, namely the rights to use, copy, modify and redistribute. The new license was named the *Open Publication License*, or OPL for short (see Appendix C).

The OPL not only protects readers from rigid copyright laws, it also has several features designed to protect authors. For example, the license stipulates that works derived from an original text must clearly be labeled as such. Modifications to a text must be identified and so must the person who made them. The original author and publisher must be acknowledged on all derivative works, and may not be used to endorse the derivative works unless they have given their explicit permission. As a further safeguard, authors may opt to invoke two additional options, which are not part of the default license configuration. The first option stipulates that redistribution of substantially modified versions is not permissible. By invoking this option the author can protect herself from "Reader's Digest" type condensed versions of her work being republished by others. The second option prohibits third parties from republishing the work or works derived from it in standard paper book format.

By invoking this option, the author can give print publishers the exclusive right to publish her work in traditional print media.

While the OPL is more restrictive than most Open Source software licenses—and this includes the Free Software Foundation's own *Free Documentation License*—it does represent a fair balance between the reader's right to use, copy, modify, and redistribute texts, and the author's need to receive credit for her work, protection of her name and professional reputation, and a chance to obtain publishing contracts with traditional print based publishers. In an academic world where print publishers still reign supreme, and tenure and promotion hinges on traditional means of publication and professional recognition, the Open Content is a bold proposition that challenges us to take a giant leap of faith.

The historical analyses of this dissertation have shown how Open Source in the software world has taught us many important lessons about the value of open and free information and the incredible achievements that can come from it. My own experiences from the enCore Open Source project reaffirm these lessons, and like David A. Wiley and many, many others, I firmly believe that they have also a lot to teach us about our own research and publishing practices in the humanities. The final challenge, therefore, is that we take this leap of faith. It is much too early to say what the application of open-source principles in academic research and publishing may lead to. Only time will answer that question. One thing we can be sure of is that the traditional rigid structures, with their copyright and intellectual property laws, will resist change every step of the way. Yet, if we believe there is a better way, we need to challenge these very structures that support the current state of affairs. I have always believed that you should practice what you preach, and that is why I am making my dissertation available

under the Open Publication License—maybe one day somebody will turn this "source code" into an encore.

Appendices

A

The enCore Manifesto

What is enCore?

enCore is a powerful online multi-user environment specifically designed for educational applications. It combines unparalleled ease of use and contemporary functionality through open standard Internet technologies and the technological power and flexibility of the MOO system (Multi-User Domain Object-Oriented). enCore is Open Source Software available at no cost. It is licensed under the Free Software Foundation's GNU GPL license. Open Source Software is freedom and empowerment!

Commitment to Education

enCore's educational mission is to provide educators with tools they can use to enhance their students' learning. The enCore team has a lot of teaching experience, which went into the design, and development of the system, and we always welcome suggestions and ideas from educators on how to improve enCore and add new useful features. Enhancing education through online environments!

Commitment to User Empowerment

enCore was designed specifically to empower administrators and users alike to make the most out of online learning. Two principal goals constitute the foundation of the enCore project. 1) Make it easy for educators to set up and run educational MOOs. 2) Make it as easy and convenient as possible for users to access and utilize the MOO technology. Give users the power to harness technology!

Commitment to the User

enCore's graphical user interface (GUI), Xpress, allows for great flexibility in the way users view and experience the MOO environment. Users who don't wish to use Xpress can use the command line interface either via Telnet, or a host of other specialized client programs. User controlled look and feel!

Commitment to Free, Open, and Flexible Solutions

enCore was built with standard internet technologies MOO, HTML, Java and Javascript. Users can easily access the system with gratis web browsers on all operating systems and platforms where such browsers are available. There is no need for special purpose client programs to use an enCore MOO. Administrators of enCore-based MOOs can easily adapt the enCore database to their specific needs and purposes. Embracing Internet standards and open solutions!

Commitment to Portability

enCore was designed to run on the standard LambdaMOO server. No additional patches are necessary for basic functionality. The enCore database is backwards

compatible with LambdaMOO without any proprietary core components. Stable and portable technology!

Commercial Use

enCore is open to commercial use under the terms specified in the GNU GPL license.

Jan Rune Holmevik and Cynthia Haynes
November 1998

B

---

enCore Portfolio

This appendix is a portfolio of enCore projects and enCore-related research, recognition, and resources between 1997 and 2003. It is intended as a showcase of the scope of the enCore outgrowth into multiple and diverse projects conducted by a wide range of individuals and institutions. It includes: 1) the official enCore online resources, including website and discussion list addresses, the list archives, and the *MOOniversity* textbook's official companion website address (site created by the author); 2) a sample list of encore MOOs (includes only currently operating MOOs as of July 2003); 3) a sample of third-party research and development grants funding enCore projects around the world; 4) a sample of third-party publications based on primary research utilizing enCore MOOs; 5) various awards and media recognition; 6) the author's Visiting Scholar appointments, invited talks, and enCore consultant work (MOO development and staff training).

enCore online resources

- enCore software distribution website:http://lingua.utdallas.edu/encore/
- enCore email discussion list: encore@utdallas.edu

- enCore email list archives: http://listar.udallas.edu/archives/encore/ (1997-present)
- enCore developers email discussion list: encore-dev@utdallas.edu
- MOOniversity companion website: http://www.abacon.com/holmevik/

Sample list of enCore MOOs

MOO Name: Lingua MOO

URL: http://lingua.utdallas.edu:7000

Location: University of Texas at Dallas

Purpose: Multipurpose educational, teaching, research

enCore version: 4.0 Beta

Contact: Jan Holmevik and Cynthia Haynes

MOO Name: MOOssiggang

URL: http://iberia.vassar.edu:7000

Location: Vassar College

Purpose: Compter-assisted Language Learning (German)

enCore version:

Contact: Jeffrey Schneider and Silke von der Emde

MOO Name: Dreistadt MOO

URL: http://cmc.uib.no:7001

Location: University of Bergen, Norway

Purpose: Compter-assisted Language Learning (German)

enCore version: 2.0.6

Contact: Daniel Jung

MOO Name: MOOlin Rouge

URL: http://cmc.uib.no:9000

Location: University of Bergen, Norway

Purpose: Compter-assisted Language Learning (French)

enCore version: 2.0.6

Contact: Daniel Jung

MOO Name: CMC MOO

URL: http://lingo.uib.no:8000/

Location: University of Bergen, Norway

Purpose: Research and development and The Midsummer Night's Dream project (Norwegian)

enCore version: 3.2

Contact: Carsten Jopp

MOO Name: OldPueblo MOO

URL: http://oldpueblomoo.arizona.edu:7000/

Location: University of Arizona

Purpose:Multipurpose educational and community

enCore version: 3.0

Contact: Jean Kreis

MOO Name: Villa Diodati (Romantic Circles MOO)

URL: http://www.rc.umd.edu:7000/

Location: University of Maryland

Purpose: Multipurpose educational

enCore version: 3.3.2

Contact: Steve Jones

MOO Name: NCTE MOO

URL: http://www.interversity.org:7000

Location: National Council of Teachers of English

Purpose: Multipurpose professional and educational

enCore version: 2.0.2

Contact: Eric Crump

MOO Name: LC_MOO (Learning Communities MOO)

URL: http://ispg.csu.edu.au:8800/

Location: Charles Sturt University, Wagga Wagga, NSW, Australia

Purpose: Multipurpose professional and educational

enCore version: 2.0.4

Contact: Ken Eustace

MOO Name: K9MOO

URL: http://ispg.csu.edu.au:9000/

Location: Charles Sturt University, Wagga Wagga, NSW, Australia

Purpose: Experimental canine campus (MOO R&D)

enCore version: 2.0.4

Contact: Ken Eustace

MOO Name: GalileoWorld (GMOO)

URL: http://ispg.csu.edu.au:51800/

Location: Charles Sturt University, Wagga Wagga, NSW, Australia

Purpose: Student project to build a galactic centre for extrasolar planets research

enCore version: 2.1.1

Contact: Ken Eustace

MOO Name: SKYMOOn

URL: http://eworks.engl.uic.edu:7000

Location: University of Illinois-Chicago

Purpose: Multipurpose educational/writing center

enCore version: 3.2

Contact: ico@uic.edu

MOO Name: MATIES MOO

URL: http://moo.sun.ac.za:7000/

Location: University of Stellenbosch, Stellenbosch, South Africa

Purpose: Multipurpose educational/Library services

enCore version:

Contact: John Dovey

MOO Name: Pro-Noun MOO

URL: http://linnell.english.purdue.edu:7000

Location: Purdue University

Purpose: Virtual Writing Environment

enCore version: 3.3.3

Contact: Erin Karper

MOO Name: ATHEMOO

URL: http://moo.hawaii.edu:7000

Location: University of Hawaii

Purpose: Theater and theater education

enCore version: 2.0.6

Contact: Juli Burk

MOO Name: NorthWoods MOO

URL: http://www.hu.mtu.edu:8000/

Location: Michigan Tech University

Purpose: Multipurpose educational

enCore version: 2.0.3

Contact: Michael Moore

MOO Name: Texas Tech English Dept MOO

URL: http://moo.engl.ttu.edu:7000/

Location: Texas Tech University

Purpose: Writing classes

enCore version:

Contact: Locke Carter

MOO Name: HowellHenry Land

URL: http://www.ehhcl.net:8889/

Location: London, England

Purpose: HHCL/Red Cell Advertising corporation MOO website for clients/staff

enCore version: 2.0 + corporate layer

Contact: Darrell Berry

MOO Name: AlaMOO

URL: http://ranger.accd.edu:7000

Location: San Antonio and Alamo Community College, San Antonio,TX

Purpose: Educational/Community

enCore version: 3.3.2

Contact: Lennie Irvin

MOO Name: Nouspace

URL: http://www.eaze.net:7000/

Location: Dallas, TX

Purpose: Multipurpose educational and professional

enCore version: 3.3.3

Contact: Dene Grigar

MOO Name: BC-MOO

URL: http://www.bridgewater.edu:7000

Location: Bridgewater College, Virginia

Purpose: Educational

enCore version: 3.3.2

Contact: Richard Bowman

MOO Name: PoeMOO

URL: http://moo.mmedu.net:8000/

Location: Stockholm University (Multimedia Pedagogik)

Purpose: Poetry/Literary (Swedish)

enCore version: 2.0.3

Contact: Claudijo Borovic

MOO Name: GNA MOO

URL: http://www.gnacademy.org:7000/

Location: Austin, TX

Purpose: Globewide Network Academy Virtual Offices

enCore version: 3.3.2

Contact: Joseph Wang

MOO Name: The SilverSea MOO

URL: http://www.cwrl.utexas.edu:9000/

Location: University of Texas at Austin

Purpose: CWRL (Computers, Writing, Research Lab)

enCore version: 3.3.2

Contact: Peg Syverson

MOO Name: Freiraum MOO

URL: http://freiraum.philo.at:7000/

Location: Vienna, Austria

Purpose: Project on Friedrich Nietzsche's Also Sprach Zarathustra (German)

enCore version: 3.3.2

Contact: Herbert Hrachovec

MOO Name: Groupe ESC Pau MOO

URL: http://moo.esc-pau.fr:7000/

Location: Pau, France

Purpose: Educational/business (French)

enCore version:

Contact: William Painter

MOO Name: EE-MOO

URL: http://ee-moo.ethz.ch

Location: Zurich, Switzerland

Purpose: Swiss Federal Institute of Technology Virtual offices

enCore version:

Contact: Tobias Oetiker

MOO Name: PROXY

URL: http://proxy.arts.uci.edu/index.html

Location: University of California-Irvine

Purpose: Experimental MOO+Proxy layer designed as critical theory/practice in the media arts

enCore version: 1.1.1

Contact: Robert Nideffer

MOO Name: AcadeMOO

URL: http://academoo.cl.msu.edu:8000/

Location: Michigan State University

Purpose: Educational/Academic Scholars Program

enCore version: 3.3.2

Contact: Tess Tavormina

MOO Name: FatecMOO

URL: http://www.fatecid.com.br:7000/

Location: Brazil

Purpose: FatecMOO is both Social and Educational, emphasis in EFL (English as a Foreign Language)

enCore version: 3.3.3

Contact: Van Souza

MOO Name: U-MOO

URL: http://umoo.uncg.edu/

Location: University of North Carolina-Greensboro

Purpose: Educational

enCore version: 3.2

Contact: Bob King

MOO Name: ASMOO

URL: http://asmoo.ipfw.edu:7000/

Location: Indiana Univ/Purdue Univ-Ft.Wayne

Purpose: Educational/Academic arts & sciences academy

enCore version: 3.2

Contact: Deb Sewards

MOO Name: G2/Lost Cities MOO

URL: http://kubrick.cdes.qut.edu.au:7000/

Location: Queensland Univ. of Technology, Australia

Purpose: Communication design technology studies

enCore version: 3.3.3

Contact: truna

MOO Name: World of Diversity MOO

URL: http://diversity.ds.psu.edu:7000/

Location: The Pennsylvania State University--DuBois

Purpose: Virtual Community created to promote diversity and encourage collaboration

enCore version: 3.2

Contact: Deborah Gill

MOO Name: TaiMOO

URL: http://TaiMOO.ntjcpa.edu.tw:7000/

Location: National Taiwan Junior College of Performing Arts, Taipei, Taiwan

Purpose: Education and English learning

enCore version: 3.3.3

Contact: Cheng-chao Su

MOO Name: JurMOO

URL: http://moo1.iig.uni-freiburg.de:7000/

Location: Freiburg, Germany

Purpose: Computer & Law teaching in a university project

enCore version: 3.2

Contact: Frank Roehr

MOO Name: enCore Italy MOO

URL: http://work.economia.unibo.it:7000

Location: University of Bologna - Italy

Purpose: Educational/Academic

enCore version: 3.3.3

Contact: Antonio Roversi

MOO Name: CLCS Campus MOO

URL: http://kontakt.tcd.ie:7000

Location: Trinity College Dublin, Ireland

Purpose: Tandem language learning partnerships involving English, German, French, Italian. CALL research and development (currently: Bilingual Tandem Analyzer)

enCore version: 3.2

Contact: Klaus Schwienhorst

MOO Name: CWRU MOO

URL: http://cwrumoo.cwru.edu:7000/

Location: Case Western Reserve University, Ohio

Purpose: General purpose educational

enCore version: 3.3.3

Contact: Guibourc

MOO Name: Seminar MOO (formerly Collegetown MOO)

URL: http://147.92.13.7:7000/

Location: Buena Vista College, Storm Lake, Iowa

Purpose: Multi-purpose educational

enCore version: 3.2

Contact: Ken Schweller

MOO Name: Ponte Italiano MOO

URL: http://www.italiano.no/

Location: University of Bergen, Norway

Purpose: Italian language learning MOO

enCore version:

Contact: Daniel Jung

MOO Name: Story MOO

URL: http://diac.it-c.dk:7000

Location: IT University of Copenhagen, Denmark

Purpose: PhD project; narrative and creative writing

enCore version: 2.1.1

Contact: Lisbeth Klastrup

MOO Name: TecfaMOO

URL: http://tecfamoo.unige.ch:7000

Location: University of Geneva, Switzerland

Purpose: General purpose educational

enCore version: 3.2

Contact: Daniel Schneider

MOO: igMOO

URL: http://hosting.uaa.alaska.edu/jeffwhite/classes/englishonline/

Location: University of Alaska-Anchorage

Purpose: General purpose educational

enCore version:

Contact: Jeff White

MOO: OakMOO

URL: http://en071.chss.iup.edu

Location: Indiana University of Pennsylvania

Purpose: English writing instruction.

enCore version: 3.2

Contact: Rob Koch

MOO: DartMOO

URL: http://dartmoo.dartmouth.edu:7000

Location: Dartmouth College

Purpose: General purpose educational

enCore version:

Contact: Mark O'Neil

MOO: schmooze MOO

URL: http://schmooze.hunter.cuny.edu:9000

Location: Hunter College, City University of New York

Purpose: ESL/EFL and general purpose educational

enCore version: 1.0

Contact: Jon Wanderer

MOO: MiamiMOO

URL: http://moo.muohio.edu:7000

Location: Miami University, Oxford, Ohio

Purpose: General purpose educational

enCore version: 3.3.3

Contact: Laura Mandell

Sample of research and development grants funding enCore-based projects

- Charles Sturt University, Wagga Wagga, Australia. "Electronic Learning Communities." Principle researcher, Ken Eustace. Information Technology and Teacher Librarianship staff of the Faculty of Science and Agriculture. (http://farrer.riv.csu.edu.au/moo/jv/) Purpose: Development of three enCore-based MOOs.
- Lingo Project, University of Bergen, Norway. "Lingo.uib." Principle researcher: Carsten Jopp. Department of Humanistic Informatics. (http://cmc.uib.no/lingo/styret/sluttrapport/) Purpose: Develop German Language MOO (Dreistadt) in Norway for teaching German as a second language to Norwegian students. Funding for this project has totaled more than 4.5 mill. Norwegian Kroner.
- Trinity College Dublin. "CLCS MOO." Principle researchers: Klaus Schwienhorst and Alexandre Borgia. CLCS Department. Grant for development of Bilingual Tandem Analyzer module for use in CLCS MOO.

- University of Arizona, Tucson. "Learning in Cyberspace: Piloting An Educational MOO for the University of Arizona." Principle researcher: Roxanne Mountford. 1998. (http://oldpueblomoo.arizona.edu:7000/2680/) Purpose: Development of OldPuebloMOO. Currently their MOO is one of three university-supported online technologies (in addition to WebCT and CAUCUS). Three of four grant-supported team members were hired by the Center for Computing and Information Technology, and one of them has received a full-time job (benefits, $35K+/yr., tuition) to be the faculty liason for the MOO (see http://oldpueblomoo.arizona.edu/awards.html for list of grants and awards).
- Vassar College. "Proposal for Mellon Foundation Program for Teaching with Technology: Utilizing a MOO for Intercultural Language Learning." Principle researchers: Jeffrey Schneider and Silke von der Emde. German Department. Purpose: Developed MOOssiggang German language MOO and conducted workshop and symposium for foreign language faculty from Vassar College and Williams College.
- Michigan Technological University. Northwoods MOO. Principle Researcher: Michael Moore. Humanities Department. Funded through "a grant from the Michigan Department of Education and funds provided through a Technology Challenge Innovation Grant totaling $1,737,940 from the U.S. Department of Education and 49% of total costs from contributions from school districts and business/industry partners in the Central Upper Peninsula of Michigan. The project is part of Project TELL -- Technology in Education through Leadership and Literacy. Over 12 school districts and institutions participate in the project.

- Texas Women's University. TWU MOO. Principle Researcher: Dene Grigar. English Department. $57,300 from Partnership Funds to develop and maintain a campus-wide interactive virtual space, called TWUMOO, for the purposes of teaching in distance learning courses and traditional educational environments, as well as for research and professional development activities. 1998-2000**.**

Sample of publications based on primary research utilizing enCore MOOs

Aarseth, Espen, and Carsten Jopp. CALLMOO fase I - Sluttrapport. 1998. http://cmc.hf.uib.no/dreistadt/eval/sluttrapp.html (6 June 2003).

Borovic, Claudijo. *What is MOO? – a cyberculture study*. Unpublished Master's thesis. Multimedia-Pedagogical Institute of Stockholm University, Sweden, forthcoming.

Donaldson, Randall P. and Markus Kötter. "Language learning in cyberspace: Teleporting the classroom into the target culture." *Calico* 16.4 1999: 531-557.

English, Joel A. *Assessing the synchronous online classroom. Methodologies and findings in real-time virtual learning environments*. Unpublished PhD dissertation. Ball State University, Department of English, 1999.

_____. "MOO-based metacognition: Incorporating online and offline reflection into the writing process." *Kairos : A Journal for Teachers of Writing in Webbed Environments*. 3.1 1998. http://english.ttu.edu/kairos/3.1/features/english/intro.html/ (17 Oct. 2002).

Grigar, Dene. "TWUMOO's Interactive Virtual Archives." *Kairos : A Journal for Teachers of Writing in Webbed Environments* 5.2 Fall 2000. http://english.ttu.edu/kairos/5.2/coverweb.html (6 June 2003).

Hammer, Anita Synnøve. *Weaving Plots: Frames of Theatre and Ritual in Simultaneous Interactive Digital Communication*. PhD Dissertation. Norwegian University of Science and Technology, 2001.

Haynes, Cynthia. "Total ReCall: Memory, MOOs, and Morphology." Keynote address at CALLMOO: Enhancing Language Learning Through Internet Technologies, University of Bergen, Norway, 1997.

_____. *WOOing Prosthetica: Dreams, Droids, and Other Feminist Technologies*. Editor of special MOO-based real-time issue of *Pre/Text: Electra(lite),* (an electronic journal of rhetorical theory), PT 2.1 Fall 1999. http://www.utdallas.edu/PT/PT2.1/ (6 June 2003).

Haynes, Cynthia and Jan Rune Holmevik, eds. *High Wired: On the Design, Use, and Theory of Educational MOOs*. (1998) Ann Arbor, MI: University of Michigan Press, 2nd edition, 2002.

Haynes, Cynthia, Jan Rune Holmevik, Beth Kolko and Victor J. Vitanza. "MOOs, Anarchitexture, Towards a New Threshold" in *The Emerging CyberCulture: Literacy, Paradigm, and Paradox*. eds. Stephanie Gibson and Ollie Oviedo. Hampton Press, 1999.

Holmevik, Jan Rune and Cynthia Haynes. *MOOniversity: A Student's Guide to Online Learning Environments*. Needham Heights, MA: Allyn & Bacon, 2000.

Karper, Erin. "Encore MOO Guide" (with Lennie Irvin) 2001, 2003. Purdue University, Writing Program. http://pw.english.purdue.edu/technologies/MOO/guide/index.html (6 June 2003).

Klastrup, Lisbeth. *Towards a Poetics of Virtual Worlds: Multi-user Textuality and the Emergence of Story*. Unpublished PhD dissertation. IT University, Copenhagen. Department of Digital Aesthetics and Communication, 2003.

Koch, Jr., Robert. *MOO Stories: A Collection of Case Studies on Teaching Writing in Object-Oriented Multi-User Domains*. Unpublished PhD dissertation. Indiana University of PA, forthcoming.

Kötter, Markus. "Networked On-line Environments: How Can They Contribute to (second) Language Acquisition?" in *Effective CALL: Wedding Theory with Practice.* eds. Randall P. Donaldson and Margaret A. Haggstrom. Netherlands: Swets and Zeitlinger, forthcoming 2003.

_____. *Tandem Learning on the Internet: Learner interactions in virtual online environments (MOOs)*. Frankfurt am Main: Peter Lang, 2002.

Love, Jane. "MOO-Scream on its wayves to WOOmb-SCREAMS." *Pre/Text Electra(Lite) 3.1 (2000)*. http://www.utdallas.edu/pretext/PT3.1/love/love.html (9 June 2003).

Nideffer, Robert. "Manufacturing Agency: Relationally Structuring Community In-Formation." *Artificial Intelligence and Society*. Spring 1999. http://time.arts.ucla.edu/AI_Society/nideffer.html (9 June 2003).

_____. PROXY. Internet Application. Java-based and enCore-based, Open Sourced Multi-Agent Management and Development Environment. (v1.0b Released Fall 2001; Ongoing Research and Development, Fall 1997-Present) http://proxy.arts.uci.edu/ (9 June, 2003).

_____. "PROXY." Exhibitions. "Biennale of Electronic Arts Perth (BEAP)." John Curtin Gallery. Perth, Australia. July 31-September 15, 2002; "Whitney Biennial." Whitney Museum of American Art. New York City, New York (http://www.whitney.org/artport/exhibitions/biennial2002/nideffer.shtml). March 7-May 26, 2002; "fusion'00." Invited Artist. Bauhaus University. Weimar, Germany. Design Media Arts. University of California, Los Angeles. June 7-9, 2000.

O'Rourke, Breffni. *Metalinguistic knowledge and instructed second language acquisition: a theoretical model and its pedagogical application in computer-mediated communication*. Unpublished PhD dissertation. Centre for Language and Communication Studies. University of Dublin, Trinity College, 2002.

Schneider, Jeff and Silke von der Emde. "Brave New (Virtual) World: Transforming language learning into cultural studies through online learning environments (MOOs)." *ADFL Bulletin* 32.1 (2000): 18-26.

Schwienhorst, Klaus. "Evaluating tandem language learning in the MOO: Discourse repair strategies in a bilingual Internet project." *CALL* 15 (2) 2002: 135-146.

_____. "Talking on the MOO: Learner autonomy and language learning in tandem." Paper presented at the CALLMOO: Enhancing Language Learning Through Internet Technologies, Bergen, Norway, 1997. http://www.tcd.ie/CLCS/assistants/kschwien/Publications/CALLMOOtalk.htm (20 Jan. 2003).

_____. *Virtual reality and learner autonomy in second language acquisition*. Unpublished PhD dissertation. Trinity College Dublin, 2000.

Tree, Everdeen. "Quick-shift." trAce Writing Center. http://trace.ntu.ac.uk/quickshift/jantext.htm (6 June 2003).

Von der Emde, Silke and Jeff Schneider. "Experiential Learning and Collaborative Reading: Literacy in the Space of Virtual Encounters." *Between the Lines: Perspectives on Foreign Language Literacy*. ed. P. Patrikis. New Haven: Yale University Press, forthcoming.

Awards and media recognition

- March 13, 1996: Lingua MOO featured in article in online version of *The Chronicle of Higher Education*. "Academic Resources on the Internet" by Jeffrey R. Young.
- Spring, 1997. *Kairos* online journal announcement. "Kairos Forms Affiliation with Renowned MOO." (http://english.ttu.edu/kairos/2.2/news/briefs/lingua.htm).
- May 28, 2000: The enCore educational MOO project won an award for "Best Rhetoric and Writing Software Product in University Education" in the Computers and Writing Conference 2000 Technology Design Competition held in Fort Worth, Texas.
- May 28, 2000: *Kairos* Award: "Best WebText of 2000," presented at Computers and Writing 2000, Fort Worth, TX, May, 2000, for Jane Love's "MOO-Scream on its wayves to WOOmb-SCREAMS," in *Pre/Text Electra(Lite) V. 3.1, 2000*.
- Oct 24, 2000: EnCore MOOs featured in article in online version of *The Chronicle of Higher Education*. "Instructors Try Out Updated MOOs as Online-Course Classrooms" by Jeffrey R. Young (http://chronicle.com/free/2000/10/2000102401u.htm).
- December 16, 2000: "DotCom Hour" interview with Tim Hoffman broadcasting out of AM 1590 WSMN in Nashua, NH. The show has over 2.5 Million potential listeners. The DotCom Hour is a regularly scheduled radio talk show that focuses on Internet & Computer related topics (http:// www.thedotcomhour.com).

- July 10, 2001: *Salon.com* article on software licensing cites Lingua MOO. (http://www.salon.com/tech/feature/2001/07/10/microsoft_school/index.html).
- November 7, 2001: UK National Grid for Learning featured Lingua MOO site as chosen by Sue Thomas of *trace Writing Community*, University of Nottingham, England (http://www.ngfl.gov.uk/features.jsp?sec=15&cat=99&res=1004).
- December 17, 2002: Norwegian national radio sent a short feature on MOO. The first part is Jan Holmevik talking about enCore MOOs, and the second part is Cynthia Haynes talking briefly about the pedagogy of MOO. The interview is in Windows Media format (http://www.nrk.no/dynasx?p_lenke_id=318995&p_artikkel_id=2377051&mswmext=.asx).

C

Open Publication License

v1.0, 8 June 1999

I. Requirements On Both Unmodified And Modified Versions

The Open Publication works may be reproduced and distributed in whole or in part, in any medium physical or electronic, provided that the terms of this license are adhered to, and that this license or an incorporation of it by reference (with any options elected by the author(s) and/or publisher) is displayed in the reproduction.

Proper form for an incorporation by reference is as follows:

> Copyright (c) <year> by <author's name or designee>. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, vX.Y or later (the latest version is presently available at http://www.opencontent.org/openpub/).

The reference must be immediately followed with any options elected by the

author(s) and/or publisher of the document (see section VI).

Commercial redistribution of Open Publication-licensed material is permitted.

Any publication in standard (paper) book form shall require the citation of the original publisher and author. The publisher and author's names shall appear on all outer surfaces of the book. On all outer surfaces of the book the original publisher's name shall be as large as the title of the work and cited as possessive with respect to the title.

II. Copyright

The copyright to each Open Publication is owned by its author(s) or designee.

III. Scope Of License

The following license terms apply to all Open Publication works, unless otherwise explicitly stated in the document.

Mere aggregation of Open Publication works or a portion of an Open Publication work with other works or programs on the same media shall not cause this license to apply to those other works. The aggregate work shall contain a notice specifying the inclusion of the Open Publication material and appropriate copyright notice.

SEVERABILITY. If any part of this license is found to be unenforceable in any jurisdiction, the remaining portions of the license remain in force.

NO WARRANTY. Open Publication works are licensed and provided "as is" without warranty of any kind, express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose or a warranty of non-infringement.

IV. Requirements On Modified Works

All modified versions of documents covered by this license, including translations, anthologies, compilations and partial documents, must meet the following requirements:

1) The modified version must be labeled as such.
2) The person making the modifications must be identified and the modifications dated.
3) Acknowledgement of the original author and publisher if applicable must be retained according to normal academic citation practices.
4) The location of the original unmodified document must be identified.
5) The original author's (or authors') name(s) may not be used to assert or imply endorsement of the resulting document without the original author's (or authors') permission.

V. Good-Practice Recommendations

In addition to the requirements of this license, it is requested from and strongly recommended of redistributors that:

1) If you are distributing Open Publication works on hardcopy or CD-ROM, you provide email notification to the authors of your intent to redistribute at least thirty days before your manuscript or media freeze, to give the authors time to provide updated documents. This notification should describe modifications, if any, made to the document.
2) All substantive modifications (including deletions) be either clearly marked up in the document or else described in an attachment to the document.
3) Finally, while it is not mandatory under this license, it is considered good form to offer a free copy of any hardcopy and CD-ROM expression of an Open Publication-licensed work to its author(s).

VI. License Options

The author(s) and/or publisher of an Open Publication-licensed document may elect certain options by appending language to the reference to or copy of the license. These options are considered part of the license instance and must be included with the license (or its incorporation by reference) in derived works.

A) To prohibit distribution of substantively modified versions without the explicit permission of the author(s). "Substantive modification" is defined as a change to the semantic content of the document, and excludes mere changes in format or typographical corrections.

   To accomplish this, add the phrase `Distribution of substantively modified versions of this document is prohibited without the explicit permission of the copyright holder.' to the license reference or copy.

B) To prohibit any publication of this work or derivative works in whole or in part in standard (paper) book form for commercial purposes is prohibited unless prior permission is obtained from the copyright holder.

To accomplish this, add the phrase 'Distribution of the work or derivative of the work in any standard (paper) book form is prohibited unless prior permission is obtained from the copyright holder.' to the license reference or copy.

References

References

Documentation style: MLA and Columbia Online Style (COS), Humanities

Aarseth, Espen. *Cybertext: Perspectives on Ergodic Literature*. Baltimore: Johns Hopkins University Press, 1997.

_____, ed. *Datahåndbok for humanister*. Oslo: Ad Notam Gyldendal, 1999.

_____. *Innføring i kulturell informatikk*. Bergen: Universitetet i Bergen, 1996.

Adams, Rich. "A History of 'Adventure'." *The Colossal Cave Adventure Page*. (7 April 2000) http://people.delphi.com/rickadams/adventure/a_history.html (17 Jan. 2001).

Allison, David K. "Smithsonian Oral and Video Histories: Marc Andreessen." Smithsonian Institution. http://americanhistory.si.edu/csr/comphist/ma1.html (29 March 2002).

Anderson, Judy. Personal Interview. 28 Jan. 2001.

Antell, Kimberly M. "Re: MUD Info." 19 April 1991. http://www.apocalypse.org/pub/u/lpb/muddex/mud-answers3.html (27 Jan. 2001).

Apache HTTP Server Version 1.3 Documentation. *Apache.org*. http://httpd.apache.org/docs/ (5 Nov. 2002).

Apache Software Foundation. http://apache.org/ (9 Nov. 2002).

Apple Computer. "The Evolution of Darwin." Apple Developer Connection. http://developer.apple.com/darwin/history.html (20 Nov. 2002).

Aspnes, James. "TinyMUD now available via telnet." 19. Aug. 1989. http://www.apocalypse.org/pub/u/lpb/muddex/a-o.html (27 Jan. 2001).

_____. Personal Interview. 27 Feb. 2001.

Aspray, William. *John Von Neumann and the Origins of Modern Computing*. Cambridge, MA.: MIT Press, 1990.

Babbage, Charles. *Passages from the Life of a Philosopher*. ed. Martin Campbell-Kelly. New Brunswick, NJ: Rutgers University Press, 1994.

Barrios, Jorge R. and Deanna Wilkes-Gibbs. "How to MOO without Making a Sound: A Guide to the Virtual Communities Known as MOOs." *High Wired: On the Design, Use, and Theory of Educational MOOs*. eds. Cynthia Haynes and Jan Rune Holmevik. (1998) Ann Arbor: University of Michigan Press, 2nd edition 2002. 45-87.

Bartle, Richard. "Incarnations of MUD." 2000. http://www.mud.co.uk/richard/incarns.htm (27 Jan. 2001).

_____. Interview. *The BL Rag*. 1997. http://www.mud.co.uk/richard/bl9c.htm (26 Jan. 2001).

_____. "Richard Bartle: Biography." 21 Jan. 1999 http://www.mud.co.uk/richard/biog.htm (16 Jan. 2001).

Berners-Lee, Tim. "C5-Conclusion." W3C. http://www.w3.org/Talks/C5_conclusion.html (28 March 2002).

_____. "Information Management: A Proposal." http://www.w3.org/History/1989/proposal.html (14 March 2002).

_____. "Re: Qualifiers on Hypertext links..." 6 Aug. 1991. News:alt.hypertext. (28 March 2002).

_____. "The World Wide Web: Past, Present and Future." http://www.w3.org/People/Berners-Lee/1996/ppf.html (13 March 2002).

Berners-Lee, Tim and Robert Cailliau. "WorldWideWeb: Proposal for a HyperText Project." http://www.w3.org/Proposal.html (27 March 2002).

Berners-Lee, Tim, Mark Fischetti, and Michael L. Dertouzos. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*. San Francisco: Harper Collins, 1999.

Boswell, David et.al. "Creating Applications with Mozilla." O'Reilly 2002. http://books.mozdev.org/ (20 Nov. 2002).

Brandt, Daniel. "PageRank: Google's Original Sin." Google-Watch.org. http://www.google-watch.org/pagerank.html (20 June 2003).

Brin, Sergey and Lawrence Page. "The Anatomy of a Large-Scale Hypertextual Web Search Engine." Seventh International World Wide Web Conference.

Brisbane Australia 14-18 April, 1998. http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm (20 June 2003)

British Legends (MUD1). Telnet://www.british-legends.com:27750 (20 Jan. 2001).

Broersma, Matthew. "Eric Raymond: Why open source will rule." ZDNet (UK) 29 March 2002. http://zdnet.com.com/2100-1104-871366.html (1 Nov. 2002).

_____. "Raymond: Mac OS X too restrictive." ZDNet (UK) 26 March 2002. http://zdnet.com.com/2100-1104-868865.html (1 Nov. 2002).

Bruckman, Amy. "Finding One's Own in Cyberspace." *High Wired: On the Design, Use, and Theory of Educational MOOs*. eds. Cynthia Haynes and Jan Rune Holmevik. (1998) Ann Arbor, MI: University of Michigan Press, 2nd edition 2002. 15-24.

_____. "MacMOOSE." http://www.cc.gatech.edu/fac/Amy.Bruckman/MacMOOSE/ (14 May 2003).

Bruckman, Amy and Mitchel Resnick. "The MediaMOO Project: Constructionism and Professional Community." *Convergence* 1:1 Spring 1995. http://asb.www.media.mit.edu/people/asb/convergence.html (12 March 2003).

Burk, Juli. "The Play's the Thing: Theatricality and the MOO Environment." *High Wired: On the Design, Use, and Theory of Educational MOOs*. eds. Cynthia

Haynes and Jan Rune Holmevik. (1998) Ann Arbor: University of Michigan Press, 2nd edition 2002. 232-49.

Burka, Lauren P. "A Hypertext History of Multi-User Dimensions." 1993. *The MUDdex*. http://www.apocalypse.org/pub/u/lpb/muddex/essay/ (26 Jan. 2001).

_____. The MUD Time Line. 1995. *The MUDdex*. http://www.apocalypse.org/pub/u/lpb/muddex/mudline.html (27 Jan. 2001).

Bush, Vannevar. "As We May Think." *The Atlantic Online*. July 1945. http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm (26 March 2002).

Campbell, Todd. "The First E-Mail Message." *PreText Magazine* 5. 1998. http://www.pretext.com/mar98/features/story2.htm (14 March 2002).

Cederqvist, Per. "Version Management with CVS." http://www.cvshome.org/docs/manual/cvs.html (4 Nov. 2002).

Ceruzzi, Paul E. *A History of Modern Computing*. Cambridge, MA.: MIT Press, 1998.

Chalmers, Rachel. "The Unknown Hackers." *Salon.com*. 17 May 2000. http://www.salon.com/tech/feature/2000/05/17/386bsd/print.html (25 Aug. 2001).

Chassell, Robert. Personal Interview. 7 Feb. 2001.

"Chat Politics." Nethistory: An Informal History of BITNET and the Internet. http://nethistory.dumbentia.com/chatpol.html (16 March 2002).

Chin, Elliot and Andrew Park. "History of Advanced Dungeons and Dragons." *Gamespot*. http://www.gamespot.com/features/history_add/ (19 Jan. 2001).

Christensen, Ward and Randy Suess. "Birth of the BBS." http://timeline.textfiles.com/1978/01/16/2/FILES/cbbs.txt (14 March 2002).

Cisco Systems Inc. "Understanding TCP/IP." http://www.cisco.com/univercd/cc/td/doc/product/iaabu/centri4/user/scf4ap1.htm (13 March 2001).

Cohen, Nancy. "Open Content: The Revolution in Publishing." http://www.open-mag.com/features/Vol_49/Perens/Perens.htm (16 June 2003).

Collegetown. Telnet://ctown.bvu.edu:7777 (17 June 2003)

Cooke, Daniel, Joseph Urban, and Scott Hamilton. "Unix and Beyond: An Interview with Ken Thompson." *Computer.org* May, 1999. http://www.computer.org/computer/thompson.htm (9 Aug. 2001).

Corbató, F. J. and Vyssotsky, V. A. "Introduction and Overview of the Multics System." Multics.org. http://www.multicians.org/fjcc1.html (8 Aug. 2001).

Crump, Eric. "At Home in the MUD: Writing Centers Learn to Wallow." *High Wired: On the Design, Use, and Theory of Educational MOOs*. eds. Cynthia

Haynes and Jan Rune Holmevik. (1998) Ann Arbor: University of Michigan Press, 2nd edition 2002. 177-91.

Curtis, Pavel. "Mudding: Social Phenomena in Text-Based Virtual Realities." (Berkeley, CA: 1992). http://www.zacha.net/articles/mudding.html (3 March 2003).

_____. "Not Just a Game: How LambdaMOO Came to Exist and What It Did to Get Back at Me." *High Wired: On the Design, Use, and Theory of Educational MOOs*. eds. Cynthia Haynes and Jan Rune Holmevik. (1998) Ann Arbor, MI: University of Michigan Press, 2nd edition 2002. 25-42.

_____. Personal Interview. 25 Jan. 2001.

Davis, D. Diane. "(Non)Fiction('s) Addiction(s): A NarcoAnalysis of Virtual Worlds." *High Wired: On the Design, Use, and Theory of Educational MOOs*. eds. Cynthia Haynes and Jan Rune Holmevik. (1998) Ann Arbor: University of Michigan Press, 2nd edition 2002. 267-85.

Day, Michael. "Re: [enCore] Research Info Needed." Personal email. (11 March 2003).

Dibble, Julian. "A Rape in Cyberspace: How an Evil Clown, a Haitian Trickster Spirit, Two Wizards, and a Cast of Dozens Turned a Database Into a Society." *Village Voice* 23 Dec. 1993. http://www.levity.com/julian/bungle_vv.html (4 Nov. 2002).

Diversity University. http://moo.du.org:8888/ (17 June 2003).

Dreistadt. http://cmc.uib.no:7001/ (20 June 2003)

"Early IRC History." The-Project.org. http://www.the-project.org/history.html (17 March 2002).

"Expanding the Universe of Ideas." *Wired News*. 17 June 1999. http://www.wired.com/news/politics/0,1283,20276,00.html (15 June 2003).

Festa, Paul. "Present at the 'e'-creation." CNET News.com. 10 Oct. 2001. http://news.com.com/2008-1082-274161.html?legacy=cnet (14 March 2002).

Fielding, Roy T. "Shared Leadership in the Apache Project." *Communications of the ACM* 42.4 (April 1999): 42-43.

Flanagan, David. *Java in a Nutshell*. Cambridge, MA.: O'Reilly and Associates 2nd edition, 1997.

Flanagan, David. *Javascript: The Definitive Guide*. Cambridge, MA.: O'Reilly and Associates 2nd edition, 1997.

Frauenheim, Ed. "Crafting the free-software future." *Salon.com* 6 March 2001. http://archive.salon.com/tech/feature/2001/03/06/sourceforge/index.html (22 Nov. 2002).

Free Software Foundation. "GNU Free Documentation License." http://www.gnu.org/licenses/fdl.html (14 Nov. 2002).

Freshmeat. "About." http://freshmeat.net/about/ (13 Nov. 2002).

_____. "Welcome to Freshmeat.net." http://freshmeat.net/ (13 Nov. 2002).

GCC. GNU Compiler Collection. http://gcc.gnu.org (29 Aug. 2001).

Giloi, Wolfgang K. "Konrad Zuse's Plankalkül: The First High-Level, 'non von Neumann' Programming Language." *Annals of the History of Computing*. 10.2 (April-June 1997): 17-24.

Glasner, Johanna. "The Biggest IPO of 1999?" *Wired News*. 18 Dec. 1999. http://www.wired.com/news/business/0,1367,33166,00.html (9 Nov. 2002).

_____. "VA Linux Sets IPO Record." *Wired News*. 9 Dec. 1999. http://www.wired.com/news/business/0,1367,33009,00.html (9 Nov. 2002).

Glave, James. "Putting a Price on Free Source." *Wired News*. 12 May 1998. http://www.wired.com/news/technology/0,1282,12262,00.html (22 Nov. 2002).

Glusman, Gustavo. "BioMOO's Purpose Statement." http://bioinfo.weizmann.ac.il/BioMOO/purpose.html (2 March 2003).

GNU. "GNU Emacs FAQ." GNU. http://www.gnu.org/software/emacs/emacs-faq.text (28 August 2001).

_____. "GNU Free Documentation License." GNU. http://www.gnu.org/licenses/fdl.html (16 June 2003).

Goldstine, Herman. *The Computer from Pascal to von Neumann*. Princeton: Princeton University Press, 1972.

Google Technology. Google http://www.google.com/technology/ (8 June 2003).

Grigar, Dene. "Dene Grigar's On-Line Dissertation Defense." Lingua MOO. 25 July 1995. http://lingua.utdallas.edu:7000/1099 (1 May 2003).

Grigar, Dene and John F. Barber. "Defending Your Life in MOOspace: A Report from the Electronic Edge." *High Wired: On the Design, Use, and Theory of Educational MOOs*. eds. Cynthia Haynes and Jan Rune Holmevik. (1998) Ann Arbor: University of Michigan Press, 2nd edition 2002. 192-31.

Guernsey, Lisa. "College "MOOs" Foster Creativity and Collaboration Among Users." *The Chronicle of Higher Education* 9 Feb. 1996. http://www.bvu.edu/ctown/CHE.html (16 March 2003).

Hafner, Katie, and Matthew Lyon. "Talking Headers." *The Washington Post Magazine*. 4 Aug. 1996. http://www.olografix.org/gubi/estate/libri/wizards/email.html (14 March 2002).

Hammel, Michael J. "The History of XFree86: Over a Decade of Development." *Linux Magazine*. Dec. 2001. http://www.linux-mag.com/2001-12/xfree86_01.html (25 March 2002).

Harold, Elliotte Rusty, and W. Scott Means. *XML in a Nutshell*. Cambridge MA.: O'Reilly and Associates 2nd edition, 2002.

Hauben, Michael, and Ronda Hauben. *Netizens: On the History and Impact of Usenet and the Internet.* http://www.columbia.edu/~rh120/ (14 March 2002).

Haynes, Cynthia. *Beta Rhetoric: Writing, Technology, and Deconstruction*. Albany, NY: State University of New York Press, forthcoming 2004.

_____. Help! There's a MOO in This Class!" *High Wired: On the Design, Use, and Theory of Educational MOOs*. eds. Cynthia Haynes and Jan Rune Holmevik. (1998) Ann Arbor: University of Michigan Press, 2nd edition 2002. 161-76.

Haynes, Cynthia and Jan Rune Holmevik, eds. *High Wired: On the Design, Use and Theory of Educational MOOs*. (1998) Ann Arbor, MI: University of Michigan Press, 2nd edition 2002.

Haynes, Cynthia, Jan Rune Holmevik, Beth Kolko, and Victor J. Vitanza. "MOOs, Anarchitexture, Towards a New Threshold." *The Emerging Cyberculture: Literacy, Paradigm, and Paradox*. eds. Stephanie Gibson and Ollie Oviedo. Cresskill, NJ: Hampton Press, 1999. 229-62.

Himanen, Pekka. *The Hacker Ethic*. New York: Random House, 2001.

Hobbler, Roy C. "Apache: More than a Web server." *ZDNet*. 30 Sept. 2002. http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2881957,00.html (25 Nov. 2002).

Hodges, Andrew. *Alan Turing: The Enigma*. New York: Walker and Company, 2000.

Holmevik, Jan Rune. *Educating the Machine. A Study in the History of Computing and the Simula Programming Languages*. Trondheim: Center for Technology and Society, 1994.

_____. "Java—programmeringsspråk som sprengjer grenser" *Datahåndbok for humanister*. ed. Espen Aarseth. Oslo: Ad Notam Gyldendal, 1999. 107-21.

_____. "Taking the MOO by the Horns: How to Design, Set Up, and Manage an Educational MOO." *High Wired: On the Design, Use, and Theory of Educational MOOs*. eds. Cynthia Haynes and Jan Rune Holmevik. (1998) Ann Arbor: University of Michigan Press, 2nd edition 2002. 107-47.

Holmevik, Jan Rune and Cynthia Haynes. "CypherText MOOves: A Dance with Real-Time Publication." *New Worlds, New Words: Exploring Pathways for Writing about and in Electronic Environments*. eds. John F. Barber and Dene Grigar. Cresskill, NJ: Hampton Press, 2001. 213-32.

_____. *MOOniversity: A Student's Guide to Online Learning Environments*. Needham Heights, MA.: Allyn & Bacon, 2000.

Hopper, Grace Murray. "Keynote Address." *History of Programming Languages*. ed. Richard Wexelblat. ACM Monograph Series (Proceedings of the History of Programming Languages Conference, Los Angeles, CA., 1978). New York: Academic Press, 1981.

Hubbard, Jordan. "A Brief History of FreeBSD." FreeBSD Documentation Project. *FreeBSD Handbook*. http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/index.html (25 Aug. 2001).

IRC Persian Gulf War Logs. http://www.ibiblio.org/pub/academic/communications/logs/Gulf-War/ (17 March 2002).

Jay's House MOO. Telnet://jhm.ccs.neu.edu:1709 (26 Jan, 2001).

Joyce, Jim. "Interview with Bill Joy." *Unix Review*. Aug. 1984. http://www.cs.pdx.edu/~kirkenda/joy84.html (23 Aug. 2001).

Joyce, Michael. "Songs of Thy Selves: Persistence, Momentariness, Recurrence and the MOO. *High Wired: On the Design, Use, and Theory of Educational MOOs*. eds. Cynthia Haynes and Jan Rune Holmevik. (1998) Ann Arbor: University of Michigan Press, 2nd edition 2002. 311-23.

Keegan, Martin. "A Classification of MUDs." *Journal of MUD Research*. 2.2 (1997). http://www.brandeis.edu/pubs/jove/HTML/v2/keegan.html (27 Jan. 2001).

Kehoe, Brendan P. *Zen and the Art of the Internet: A Beginner's Guide to the Internet*. (1992) http://www.cs.indiana.edu/docproject/zen/zen-1.0_toc.html (17 March 2002).

Kell, Jeff. "Relay: Past, Present, and Future." NetHistory.com. Excerpt from presentation at NETCON, New Orleans, 1987. http://nethistory.dumbentia.com/relayhist.html (16 March 2002).

Kernighan, Brian and Dennis Ritchie. *The C Programming Language.* Englewood Cliffs, NJ: Prentice Hall, 1978.

Kolko, Beth. "Bodies in Place: Real Politics, Real Pedagogy, and Virtual Space." *High Wired: On the Design, Use, and Theory of Educational MOOs*. eds. Cynthia Haynes and Jan Rune Holmevik. (1998) Ann Arbor: University of Michigan Press, 2nd edition 2002. 253-65.

Kornblum, Janet. "Netscape sets source code free." *c|net*. 31 March 1998. http://news.com.com/2100-1001-209666.html?legacy=cnet (20 Nov. 2002).

Lambda MOO. Telnet://lambda.moo.mud.org:8000 (26 Jan. 2001).

Landow, George P. *Hypertext: The Convergence of Contemporary Critical Theory and Technology*. Baltimore: Johns Hopkins University Press, 1992.

Lash, Alex. "Netscape's enterprise challenge." *C|net* 26 Nov. 1997. http://news.com.com/2100-1001-205794.html?tag=rn (3 Nov. 2002).

Lee, J.A.N. "Charles Babbage." Sept. 1994. Department of Computer Science, Virginia Tech University. http://ei.cs.vt.edu/~history/Babbage.html (25 Nov. 2002).

_____. "Konrad Zuse." Sept. 1994. Department of Computer Science, Virginia Tech University. http://ei.cs.vt.edu/~history/Zuse.html (25 Nov. 2002).

Leiner, Barry M. et al. "A Brief History of the Internet." Internet Society (ISOC). http://www.isoc.org/internet-history/brief.html (16 Jan. 2002).

Leonard, Andrew. "BSD Unix: Power to the people from the code." *Salon.com*. 16 May 2000. http://www.salon.com/tech/fsp/2000/05/16/chapter_2_part_one/print.html (23 Aug. 2001).

_____. "Let my software go!" *Salon.com.* 30 March 1998. http://dir.salon.com/21st/feature/1998/04/cov_14feature.html (17 Nov. 2002).

_____. "Mozilla dreams." *Salon.com*. 10 Feb. 2000. http://dir.salon.com/tech/feature/2000/02/10/mozilla/index.html (17 Nov. 2002).

_____. "Mozilla's revenge." *Salon.com.* 12 March 2002. http://www.salon.com/tech/col/leon/2002/03/12/mozilla/index.html (17 Nov. 2002).

_____. "The Richard Stallman saga, redux." *Salon.com*. 11 Sept. 1998. http://dir.salon.com/21st/feature/1998/09/11feature.html (18 Nov. 2002).

_____. "The saint of free software." *Salon.com*. 30 July 1998. http://dir.salon.com/21st/feature/1998/08/cov_31feature.html (2 Nov. 1998).

_____. "The shape of open source to come." *Salon.com.* 3 Feb. 2000. http://dir.salon.com/tech/log/2000/02/03/slashdot/index.html (19 Nov. 2002).

Levy, Stephen. *Hackers: Heroes of the Computer Revolution*. New York: Penguin, 1984.

Lingo.uib. http://lingo.uib.no/lingo-uib/ (14 May 2003).

Lingua MOO. http://lingua.utdallas.edu:7000 (25 Jan. 2001).

Linus vs. Tanenbaum. 29 Jan. 1992. http://mm.iit.uni-miskolc.hu/Data/texts/linus_vs_tanenbaum.html (16 June 2003).

Malda, Rob. "About Rob Malda." http://www.cmdrtaco.net/rob.shtml (18 Nov. 2002).

Manjoo, Farhad. "Mozilla rising." *Salon.com.* 10 Sept. 2002. http://www.salon.com/tech/feature/2002/09/10/browser_wars/ (26 Nov. 2002).

Markoff, John and Amy Harmon. "Internal Memo Shows Microsoft Executives' Concern Over Free Software." *New York Times*. 3 Nov. 1998. http://www.nytimes.com/library/tech/98/11/biztech/articles/03memo.html (3 Nov. 2002).

McCarthy, John. "A Time Sharing Operator Program for our Projected IBM 709." 1 Jan. 1959. http://www-formal.stanford.edu/jmc/history/timesharing-memo/timesharing-memo.html (31 July 2001).

_____. "Reminiscences on the History of Time Sharing." 1983. http://www-formal.stanford.edu/jmc/history/timesharing/timesharing.html (30 July 2001).

McCool, Rob, Roy T. Fielding and Brian Behlendorf. "The Apache Story." *Linux Magazine* June 1999. http://www.linux-mag.com/1999-06/apache_01.html (2 Nov. 2002).

McHugh, Josh. "For the love of Hacking." *Forbes Magazine*. 10 Aug. 1998. http://www.forbes.com/forbes/1998/0810/6203094a.html (13 Nov. 2002).

McKusick, Marshall Kirk. "Twenty Years of Berkeley Unix: From AT&T-Owned to Freely Redistributable." *Open Sources: Voices from the Open Source*

Revolution. eds. Chris DiBona, Sam Ockman and Mark Stone. O'Reilly, 1999. http://www.oreilly.com/catalog/opensources/book/kirkmck.html (16 Aug. 2001).

McMillan Robert. "Our Man in Palo Alto." *Linux Magazine* Sept. 2001. http://www.linux-mag.com/2001-09/perens_01.html (17 Nov. 2002).

MediaMOO. telnet://mediamoo.engl.niu.edu 8888 (17 June 2003).

Meyer, Eric. *Cascading Style Sheets: The Definitive Guide*. Cambridge MA.: O'Reilly and Associates, 2000.

Microsoft. "Linux and the Open Software Source Model - A Question and Answer Session With Ed Muth Enterprise Marketing Group Manager, Microsoft Corp." Microsoft. http://web.archive.org/web/20010211142733/http://www.microsoft.com/ntserver/nts/news/mwarv/linuxresp.asp (21 Nov. 2002).

Mockus, Audris, Roy T, Fielding and James D. Herbsleb. "Two Case Studies of Open Source Software Development: Apache and Mozilla." http://www.ics.uci.edu/~wscacchi/Software-Process/Readings/mozilla-apache-case-study-TOSEM-2002.pdf (3 Nov. 2002).

"Mozilla 1.0 Release Announcement." *Mozilla.org*. 5 June 2002. http://www.mozilla.org/releases/mozilla1.0.html (11 Nov. 2002).

Mozilla. "Development Roadmap." *Mozilla.org*. 26 Oct. 1998. http://www.mozilla.org/roadmap/roadmap-26-Oct-1998.html (11 Nov. 2002).

Mozilla. “nglayout project / gecko layout engine.” *Mozilla.org.* 7 Nov. 2000. http://www.mozilla.org/newlayout/faq.html (11 Nov. 2002).

Mozilla. “Our Mission.” *Mozilla.org*. http://www.mozilla.org/mission.html (11 Nov. 2002).

Mozilla. “Releases.” *Mozilla.org*. http://www.mozilla.org/releases/ (11 Nov. 2002).

Muffett, Alec. “The AberMUD1 History Site.” http://www.users.dircon.co.uk/~crypto/abermud/amud.html (28 Jan. 2001).

“Multics History.” Multicians.org. 31 October 2000. http://www.multicians.org/history.html (8 Aug. 2001).

Musciano, Chuck and Bill Kennedy. *HTML & XHTML: The Definitive Guide*. Cambridge MA.: O’Reilly and Associates 4th edition, 2000.

Naur, Peter and Brian Randell. “Software Engineering: Report on a conference sponsored by the NATO Science Committee.” Garmish, Germany. 7-11 Oct. 1968. http://homepages.cs.ncl.ac.uk/brian.randell/NATO/ (3 Nov. 2002).

NCSA. “About NCSA Mosaic for X.” http://archive.ncsa.uiuc.edu/SDG/Experimental/demoweb/old/mosaic-docs/help-about.html (29 March 2002).

“NCSA HTTPd.” *NCSA*. http://hoohoo.ncsa.uiuc.edu/index.html (2 Nov. 2002).

NetBSD Project. http://www.netbsd.org/ (25 Aug. 2001).

"Netcraft Web Server Survey." Netcraft. Oct. 2002. http://www.netcraft.com/survey/ (15 Nov. 2002).

Netscape. "America Online, Inc. Completes Acquisition of Netscape Communications Corporation." 17 March 1999. http://wp.netscape.com/newsref/pr/newsrelease746.html (18 Nov. 2002).

_____. "Netscape Accelerates Communicator Evolution With First Release Of Next-Generation Communicator Source Code To Developer Community Via Mozilla.Org." 31 March 1998. http://wp.netscape.com/newsref/pr/newsrelease591.html (18 Nov. 2002).

_____. "Netscape Announces Plans To Make Next-Generation Communicator Source Code Available Free On The Net." 22 Jan. 1998. http://wp.netscape.com/newsref/pr/newsrelease558.html (18 Nov. 2002).

_____. "Netscape Makes Draft of Free Source Code License Available for Review." 4 March 1998. http://wp.netscape.com/newsref/pr/newsrelease579.html (18 Nov. 2002).

Norberg, Arthur L., Judy E. O'Neill and Kerry J. Freedman. *Transforming Computer Technology : Information Processing for the Pentagon, 1962-1986*. Baltimore: Johns Hopkins University Press, 1996.

Nussbacher, Henry. "Chat Analysis: A Tour-De-Force Analysis of message and file buffers in an RSCS network." Nethistory.com. http://nethistory.dumbentia.com/chatan.html (6 March 2002).

Oakes, Chris. "Mozilla's First Birthday." *Wired News.* 1 April 1999. http://www.wired.com/news/technology/0,1282,18896,00.html (25 Nov. 2002).

_____. "MS: Open Source is Direct Threat." *Wired News.* 2 Nov. 1998. http://www.wired.com/news/technology/0,1282,15990,00.html (25 Nov. 2002).

Oeschger, Ian and David Boswell. "Getting Your Work Into Mozilla." *O'Reilly Network*. 9 Sept. 2000. http://www.oreillynet.com/pub/a/mozilla/2000/09/29/keys.html (14 Nov. 2002).

Oikarinen, Jarkko. "IRC History." *IRC.org.* http://www.irc.org/history_docs/jarkko.html (16 March 2002).

Olsen, Stefanie. "Does search engine's power threaten Web's independence?" *CNET News*. 31 Oct. 2002. http://news.com.com/2009-1023-963618.html (20 June 2003).

Open Content. "Open Publication License." *Opencontent.org.* Draft v 1.0. 8 June 1999. http://www.opencontent.org/openpub/ (24 Nov. 2002).

Open Source Initiative. "History of the OSI." http://www.opensource.org/docs/history.php (24 Nov. 2002).

_____. "The Open Source Definition." Version 1.9. http://www.opensource.org/docs/definition.php (24 Nov. 2002).

Patrizio, Andy. "Unix Growth Still Outpaces Win NT." *TechWeb News*. 30 Oct. 1998. http://www.techweb.com/wire/story/TWB19981029S0001 (4 Nov. 2002).

Perens, Bruce. "It's time to talk about Free Software again." Email. *Slashdot.org*. 18 Feb. 1999. http://slashdot.org/articles/99/02/18/0927202.shtml (29 Nov. 2002).

Raggett, Dave, Janny Lam, Ian Alexander and Michael Kmiec. *Raggett on HTML 4*. Essex, England: Addison Wesley Longman, 1998. http://www.w3.org/People/Raggett/book4/ch02.html (March 28, 2002).

Raymond, Eric S. "A Brief History of Hackerdom." Version 1.24. http://www.tuxedo.org/~esr/writings/homesteading/hacker-history/ (19 Nov. 2002).

_____. "The Cathedral and the Bazaar." Version 3.0. http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ (19 Nov. 2002).

_____. "The Halloween Documents." http://www.opensource.org/halloween/ (19 Nov. 2002).

_____. "Halloween I: 'Open Source Software: A (New?) Development Methodology?' by Vindod Valloppillil." *Microsoft*. Version 1.14 annotated by Eric S. Raymond. http://www.opensource.org/halloween/halloween1.php (5 Nov. 2002).

_____. "Halloween II: 'Linux OS Competitive Analysis: The Next Java VM?' by Vinod Valloppillil, and Josh Cohen." *Microsoft*. Version 1.4 annotated by Eric S. Raymond. http://www.opensource.org/halloween/halloween2.php (5 Nov. 2002).

_____. "Halloween IV: When Software Things Were Rotten." http://www.opensource.org/halloween/halloween4.php (5 Nov. 2002).

_____. "Homesteading the Noosphere." Version 3.0. http://www.tuxedo.org/~esr/writings/homesteading/homesteading/ (19 Nov. 2002).

_____. "How To Become a Hacker." Version 1.15. http://www.tuxedo.org/~esr/faqs/hacker-howto.html (19 Nov. 2002).

_____. "The Magic Cauldron." Version 3.0. http://www.tuxedo.org/~esr/writings/homesteading/magic-cauldron/ (19 Nov. 2002).

_____. "The New Hacker's Dictionary." http://www.tuxedo.org/~esr/jargon/jargon.html (19 Nov. 2002).

_____. Personal Interview. 9 Feb. 2001.

_____. "The Revenge of the Hackers." *Open Sources: Voices from the Open Source Revolution*. eds. Chris DiBona, Sam Ockman and Mark Stone. O'Reilly, 1999. http://www.oreilly.com/catalog/opensources/book/raymond2.html (3 Nov. 2002).

Reddick, Randy and Elliot King. *The Online Student: Making the Grade on the Internet*. Fort Worth, TX: Harcourt Brace, 1996.

Redhat Linux. "What is Red Hat Linux?" Redhat.com. http://www.redhat.com/about/whatis_rhl.html (12 Nov. 2002).

Redmond, Kent C. and Thomas M. Smith. *Project Whirlwind: The History of a Pioneer Computer*. Bedford, MA.: Digital Press, 1980.

Rheingold, Howard. *Tools for Thought*. Prentice Hall. 1986. http://www.rheingold.com/texts/tft/ (14 Nov. 2002).

_____. *The Virtual Community: Homesteading on the Electronic Frontier*. New York: Harper Perennial, 1994.

Risan, Lars. "Hackers Produce More Than Software, They Produce Hackers." http://folk.uio.no/lrisan/Linux/Identity_games/ (29 Nov. 2002).

Ritchie, Dennis. "The Evolution of the Unix Time-sharing System." http://cm.bell-labs.com/cm/cs/who/dmr/hist.html (9 Aug. 2001).

_____. "The Unix Time-sharing System—A Retrospective." http://cm.bell-labs.com/cm/cs/who/dmr/retro.html (17 Aug. 2001).

Ritchie, Dennis and Thompson, Ken. "The UNIX Time-Sharing System." http://cm.bell-labs.com/cm/cs/who/dmr/cacm.html (10 Aug. 2001).

Robbins, Arnold. *UNIX in a Nutshell: System V Edition*. Cambridge MA.: O'Reilly and Associates, 1999.

Rosenberg, Scott. "Let's Get This Straight: Microsoft's Halloween Scare." *Salon.com*. 4 Nov. 1998. http://dir.salon.com/21st/rose/1998/11/04straight.html (11 Nov. 2002).

Sammet, Jean E. *Programming Languages: History and Fundamentals*. Englewood Cliffs, NJ: Prentice-Hall, 1969.

Schweller, Ken. "MOO Educational Tools." *High Wired: On the Design, Use, and Theory of Educational MOOs*. eds. Cynthia Haynes and Jan Rune Holmevik. (1998) Ann Arbor: University of Michigan Press, 2nd edition 2002. 88-106.

SCO. "History of SCO." SCO. http://www.scoinc.com/about/history.html (25 Aug. 2001).

Shields, Rob, ed. *Cultures of Internet: Virtual Spaces, Real Histories, Living Bodies*. London: Sage Publications, 1996.

Slashdot. "Slashdot FAQ." Slashdot. http://slashdot.org/faq/ (25 Nov. 2002).

_____. "Slashdot: News for nerds, stuff that matters." Slashdot. http://slashdot.org/ (25 Nov. 2002).

_____. "SLASH: The Slashdot Code." Slashdot. http://slashdot.org/code.shtml (25 Nov. 2002).

SourceForge. "About SourceForge.net." Sourceforge. http://sourceforge.net/docman/display_doc.php?docid=6025&group_id=1 (20 Nov. 2002).

_____. "Welcome." Sourceforge. http://sourceforge.net/ (20 Nov. 2002).

Spufford, Francis and Jenny Uglow, eds. *Cultural Babbage: Technology, Time, and Invention*. London: Faber and Faber, 1996.

Stallman, Richard M. "Apple's non-free source license." *Linux Today.* 22 March 1999. http://linuxtoday.com/news_story.php3?ltsn=1999-03-22-001-05-NW-LF (2 Nov. 2002).

_____. "The Free Software Definition." GNU.org. http://www.gnu.org/philosophy/free-sw.html (18 Nov. 2002).

_____. *Free Software, Free Society: Selected Essays of Richard M. Stallman*. ed. Joshua Gay. Boston: MA.: Free Software Foundation, 2002.

_____. "The GNU Operating System and the Free Software Movement." *Open Sources: Voices from the Open Source Revolution*. eds. Chris DiBona, Sam Ockman and Mark Stone. O'Reilly. 1999. http://www.oreilly.com/catalog/opensources/book/stallman.html (28 Aug. 2001).

_____. "Initial Announcement." GNU.org. http://www.gnu.org/gnu/initial-announcement.html (28 Aug. 2001).

_____. "Lecture at KTH." Royal Institute of Technology. Stockholm, Sweden. 30 Oct. 1986. http://www.fsf.org/philosophy/stallman-kth.html (30 Nov. 2002).

_____. "Why 'Free Software' is better than 'Open Source'." GNU.org. http://www.gnu.org/philosophy/free-software-for-freedom.html (15 Nov. 2002).

Stallman, Richard M. and Bradley M. Kuhn. "Freedom or Power?" GNU.org. http://www.gnu.org/philosophy/freedom-or-power.html (15 Nov. 2002).

Staudenmaier, John M. *Technology's Storytellers: Reweaving the Human Fabric*. Cambridge, MA.: MIT Press, 1989.

Stern, Nancy. *From ENIAC to UNIVAC*. Bedford, MA.: Digital Press, 1981.

Stroustrup, Bjarne. "A History of C++: 1979-1991." *SIGPLAN Notices*. 28.3 March 1993, 271-297.

Suárez-Potts, Lous. "Interview: Frank Hecker." OpenOffice.org. 1 May 2001. http://www.openoffice.org/editorial/ec1May.html (29 Nov. 2002).

Tanenbaum, Andrew. Personal FAQ. http://www.cs.vu.nl/~ast/home/faq.html (30 Aug. 2001).

"Testimony from Netscape CEO James Barksdale." *Time.com*. http://www.time.com/time/daily/special/microsoft/documents/barksdale/index.html (5 Nov. 2002).

Toole, Betty A., ed. *Ada, The Enchantress of Numbers: A Selection from the Letters of Lord Byron's Daughter and Her Description of the First Computer*. CarTech, Inc., 1998.

Torvalds, Linus. "Linux History." http://www.li.org/linuxhistory.php (30 Aug. 2001).

_____. "The Linux Edge." *Open Sources: Voices from the Open Source Revolution*. eds. Chris DiBona, Sam Ockman and Mark Stone. O'Reilly, 1999.

http://www.oreilly.com/catalog/opensources/book/linus.html (30 Aug. 2001).

Torvalds, Linus and David Diamond. *Just for Fun: The Story of an Accidental Revolutionary*. New York: Harper Business, 2001.

Toth, Viktor T. "MUD1 History." MUD. 20 Sept. 2000. http://www.british-legends.com/history.htm (27 Jan. 2001).

Turing, Alan M. "On computable numbers: With an application to the Entscheidungsproblem." *Proceedings of the London Mathematical Society*. ser. 2, 42. (1936-37). London Mathematical Society. 230-265.

Turkle, Sherry. "All MOOs are Educational—the Experience of 'Walking through the Self'." *High Wired: On the Design, Use, and Theory of Educational MOOs*. eds. Cynthia Haynes and Jan Rune Holmevik. Ann Arbor: University of Michigan Press, 1998. ix-xix.

_____. *Life on the Screen: Identity in the Age of the Internet*. New York: Simon & Schuster, 1995.

Undernet Public Relations Committee. "Interview Log with Jarkko Oikarinen." http://www.mirc.co.uk/help/jarkko2.txt (17 March 2002).

Vitanza, Victor. "Of MOOs, Folds, and Non-reactionary Virtual Communities." *High Wired: On the Design, Use, and Theory of Educational MOOs*. eds. Cynthia Haynes and Jan Rune Holmevik. (1998) Ann Arbor, MI: University of Michigan Press, 2nd edition 2002. 286-310.

Von Neumann, John. "First Draft of a Report on the EDVAC." Moore School of Electrical Engineering. University of Pennsylvania. 30 June 1945. http://www.histech.rwth-aachen.de/www/quellen/vnedvac.pdf (20 Nov. 2002).

W3C. "A Little History of the World Wide Web." http://www.w3.org/History.html (27 March 2002).

Welsh, Matt. "Linux Installation and Getting Started." Ver. 1. 5. Aug. 1993. http://www.cs.indiana.edu/usr/local/www/linux/gs/gs.html (1 Aug. 2001).

Wexelblat, Richard L. *History of Programming Languages*. ACM Monograph Series. (Proceedings of the History of Programming Languages Conference, Los Angeles, CA., 1978). New York: Academic Press, 1981.

White, Stephen. "Moospec.txt." Apocalypse.org. 2 May 1990. http://www.apocalypse.org/pub/u/lpb/muddex/moospec.txt (28 Jan. 2001).

Wilbur, Shawn P. "Day-to-Day MOO Administration and How to Survive It." *High Wired: On the Design, Use, and Theory of Educational MOOs*. eds. Cynthia Haynes and Jan Rune Holmevik. (1998) Ann Arbor: University of Michigan Press, 2nd edition 2002. 148-58.

Wiley, David A. "Open Publication License." http://opencontent.org/openpub/ (16 June 2003).

Williams, Michael R. *A History of Computing Technology*. California: IEEE Computer Society Press, 1997.

Young, Robert. "Giving It Away: How Red Hat Software Stumbled Across a New Economic Model and Helped Improve an Industry." *Open Sources: Voices from the Open Source Revolution*. eds. Chris DiBona, Sam Ockman and Mark Stone. O'Reilly's, 1999. http://www.oreilly.com/catalog/opensources/ (4 Nov. 2002).

Zawinski, Jamie. "Fear and loathing on the merger trail." *Mozilla.org*. 23 Nov. 1998. http://www.mozilla.org/fear.html (2 Nov. 2002).

_____. "The Netscape Dorm." http://www.jwz.org/gruntle/nscpdorm.html (2 Nov. 2002).

_____. "Nomo zilla: Resignation and Postmortem." 31 March 1999. http://www.jwz.org/gruntle/nomo.html (2 Nov. 2002).