Preface

SOFTWARE IN THE MODERN CONTEXT

In many ways, software has become the life's blood of the modern world. Software allows businesses to compile and share data—literally—at the speed of light. Software also permits governments and other public sector organizations to oversee numerous activities, administer vast territories, and analyze developing situations. So powerful is the hold software has over the modern world that the fear of a global software crash in the form of the *millennium bug* led many to associate widespread software glitches with the end of civilization as we know it.

While the turn of the millennium passed without major incident, the importance of software in society continues to grow. This time, however, individuals and organizations are increasingly turning their attention from software use in industrialized nations to computing practices and online access in the developing world. As a result, concerns over the millennium bug have given way to growing interest in the global digital divide. And public and private organizations alike are increasingly examining how computers, online access, and software might help developing nations advance economically, politically, and socially.

The expanding and interlinked nature of global software use presents new situations and raises new questions for organizations and individuals alike. For example, what kinds of software should be used when and how? What are the economic, social, and political ramifications of deciding to use one kind of software instead of another? Likewise, choices based on responses to these questions can affect interaction and integration on both local and global scales. For these reasons, it is now more important than ever that individuals have access to the information needed to make informed choices about software adoption and use.

This edited collection is an initial step toward providing some of the information needed to make such informed choices. By presenting readers with a range of perspectives on a particular kind of software—open source software (OSS)—the editors believe they can shed light on some of the issues affecting the complex nature of software use in different contexts. Moreover, by bringing together the ideas and opinions of researchers, scholars, businesspersons, and programmers from six continents and 20 nations, the editors believe this collection can help readers appreciate the effects of software-related decisions in today's interconnected global environment.

OPEN SOURCE SOFTWARE (OSS): AN OVERVIEW

Software is essentially programming code—or *source code*—that provides a computer's operating system with instructions on how to perform certain tasks. The source code of a word processing program, for example, provides an operating system with information on how to translate certain keystrokes into specific text that appears on digital (and later print) pages. In essence, if a person knows the source code needed to make a computer perform a particular operation, then that individual can simply enter such source code into his or her operating system, and the computer will respond as desired. Such a response, moreover, would be the same one the computer would provide in relation to the original software product.

Within this framework, if an individual can access the underlying source code for a software product, then that person can just copy the code and not need to purchase the original program. For this reason, many software companies *close off* access to the source coding that allows their programs to work. As a result, users cannot readily access and copy the *mechanism* that makes the product work (and thus, gives it value). Instead, individuals must use an interface that allows them to activate certain commands indirectly within a software program's underlying source code. Such closed programs are know as *proprietary software*, for only the creator/copyright holder of that software is allowed to open or to see and to copy or manipulate the underlying source code.

Open source software (OSS), however, represents a completely opposite perspective in terms of access to source code. OSS products are, in essence, created in such a way that access to the underlying source code is *open* and available for others to access and review. A very basic yet common example of such open access to coding is HTML, which allows browsers to display Web pages. The coding of these pages is generally open for anyone to review and replicate. All one needs to do is access a page's underlying coding by using the "View Source"—or other related option—in his or her browser.

Such openness means individuals do not need to buy open source software in order to use it. Rather, they can review and copy a program's underlying source code and thus create a "free" version of the related software. Such openness also means individuals can modify source code and thus alter or enhance the related program. So, in theory, the foundational source code of one software product could be modified to perform different functions—or even become an entirely new program. Updating software, in turn, becomes a matter of copying new or modified code vs. purchasing the newest version of a product.

From both a local and an international perspective, OSS can provide individuals with access to affordable software that allows them both to engage in computing activities and to access others via online media. Moreover, the flexibility of OSS means individuals can modify the software they use to perform a wide variety of tasks. Doing so reduces the need for buying different programs in order to perform different activities. Thus, it is perhaps no surprise that the use of OSS is growing rapidly in many of the world's developing nations where the costs of proprietary software is often prohibitive for many citizens.

LIMITATIONS OF OPEN SOURCE SOFTWARE: A BASIC PERSPECTIVE

The easily accessible and flexible nature of open source software makes it ideal to use in a variety of contexts. OSS, however, also brings with it certain limitations that could affect interactions among individuals. Many of these limitations, in turn, arise from the fact that OSS is often developed and supported by communities of *volunteer* programmers who create and modify items in their free time.

First, and perhaps foremost, because OSS is open for the user to modify as desired, it is easy for each individual to use the same programming foundation/source code to develop different non-compatible softwares. Such divergence is often referred to as *forking code*. In such cases, each programmer working on the development of an OSS item can take a different "fork" in the programming *road*, and with each different fork, two programs that were once identical become increasingly different from one another.

Such forking code, moreover, has long been considered a major problem in OSS development.

These prospects for divergence mean OSS use is open to a variety of problems involving compatibility. Such problems include:

- Individuals generating software that others cannot use due to compatibility issues
- Software that does not work as desired or work in unexpected ways
- Parts of distributed programming projects not working as intended or not working at all
- Users becoming frustrated with and abandoning software they consider too time-consuming or cumbersome to operate

Thus, the freedom that allows one individual to operate software might prevent others from making use of such materials. As a result, access to and exchanges among others—perhaps the central focus of many of today's software packages—are impeded by the software itself.

Some companies, such as Linux, have addressed the problem of forking code and compatibility through focused oversight processes that govern programming practices. The result has been successful and relatively stable software products that work effectively with other systems. The same kind of management, oversight, and standardization, however, becomes more complicated in most OSS development/production situations where the standard approach often involves a group of globally dispersed, unpaid individuals creating OSS products in their spare time.

A second major problem area for OSS involves the technical support available to users of such software. Because it is often the case that no individual or organization really *owns* an open source software product, there is often no formal or standard mechanism for providing technical support to OSS users. Such support instead tends to come from loose networks of OSS developers and aficionados who interact informally in online contexts such as chat rooms or listserevs. Within this context, *technical support* generally means a user who is experiencing difficulty posts a query to an online OSS forum and then waits for a member of that forum to read the posting and reply.

One limitation of such an informal support system is that answers are not readily available. Instead, individuals could find themselves waiting for anywhere from seconds to days from some random community member to respond. Such delays could, in turn, have a major effect on the usability and the desirability of OSS products—not to mention the successes with which individuals can use such products to interact. Equally problematic is that such technical support systems are open for anyone to participate in and provide advice or solutions—regardless of the technical skills of the individual. Thus, the quality of the advice provide by OSS support systems can be haphazard, inconsistent, or even incorrect.

While these are but two problem areas, they illustrate the complexities bound up in selecting and using software effectively. Moreover, the use of OSS vs. proprietary software becomes increasingly intertwined with different social and political perspectives related to computing use. As a result, software choices can be as much a matter of socio-political ideology as they can be about using a product to perform a task.

SOCIAL PERSPECTIVES ON OSS: OWNERSHIP AND ECONOMICS

Much of the software we use today is proprietary. In other words, the source code that makes it work is not accessible for modification by those that purchase it. Notable opponents of proprietary software, such as the creator of the GNU operating system, Richard Stallman, led efforts to develop and distribute software and its source code freely. These efforts became known as the free software movement (FSF) and enabled software developers to both use and modify FSF items. The only stipulation these initial programmers imposed was that individuals who used FSF as a foundation for developing other items needed to make such modified code freely available. Perhaps the most successful example of FSF is the GNU/Linux operating system, which continues to increase its market share in direct competition with Microsoft's proprietary systems.

As attractive as Stallman's free software approach was to many, it also alienated others who believed that such a revolutionary and also intransigent stand—one that insisted all software code be made freely available—was not feasible. Additionally, this socio-political stand to programming was often confusing—particularly to individuals from outside of this community of programmers, for many of them interpreted the word *free* to mean nothing could be charged for and no profit could be made from developing such software. As a result of this perception, many entrepreneurial developers and software companies refrained from participating more actively in supporting Linux and other FSF products. Because of this view, a number of FSF developers, including Eric Raymond and Bruce Perens, met in 1998 to put a more business-friendly face on free software. What they eventually developed was the concept of open source software (OSS). Unlike free software, OSS was more flexible and even offered additional licensing possibilities that allow individuals to mix proprietary and open software. One example of this "hybrid" approach is the Berkeley Distribution License, which allows software developers to modify source code and then take their modifications private, even selling them for a profit, rather than having to make them freely available to others, including competitors.

PRAGMATIC APPLICATIONS: EXPLORING OPTIONS FOR OSS USE

For all of the inroads OSS has made in encouraging businesses to adopt it, it still has its detractors who rightly point out that—as noted earlier—most OSS is produced by volunteer hobbyists with no financial incentive to contribute to or continue supporting products they've produced. OSS also suffers from customer service problems since no one company necessarily exists to stand behind a product. In addition, with developers free to modify source code and generally distribute it however they wish, many different derivatives of the same basic software (*forking code*) can exist, leading to confusion and incompatibility. Finally, because of its ever-increasing popularity, OSS is something more businesses and more developers are interested in leveraging or contributing to, resulting in a market that some argue supports too many products and too few developers to go around supporting them all. For example, although there were only a handful of OSS content management systems just a few years ago, there are now scores of such systems—a situation that makes it difficult for consumers to decide which one to use.

Despite such negatives, OSS is not a passing fancy. In fact, many organizations have followed the lead of RedHat (a distributor and service supporter of Linux) in exploring ways to develop business models that maximize the advantages of OSS while maintaining the openness and flexibility of products. IBM, for example, commits a large part of its core group of developers to building and/or enhancing OSS. In addition, organizations and even governmental bodies, ranging from small not-for-profits to the European Union, have actually adopted OSS for use, with even more exploring how OSS can contribute to their core operations. Understanding how to do this, considering that OSS is a relatively new player, is challenging. It requires knowledge not only of the origins and the operating principles of OSS, but also knowledge of the social, legal, and economic factors that affect the use of OSS products.

SEARCHING FOR UNDERSTANDING WITHIN THE COMPLEXITY: OBJECTIVE AND ORGANIZATION OF THIS HANDBOOK

The decision to purchase or to use a particular software product is an important one that can contribute to the success or the failure of the related organization. For this reason, decision makers at different levels and in a variety of fields need to familiarize themselves with the various factors that contribute to the successful adoption and use of software products. Similarly, individuals need to make better-informed choices about what software to select or personal use and why. In the case of open source software, such decisions are further complicated by the social agendas and economic goals many developers and users attach to the use of OSS materials.

The objective of this handbook is to provide readers with a foundational understanding of the origins, operating principles, legalities, social factors, and economic forces that affect the uses of open source software. To achieve this objective, this handbook is divided into seven major sections, each of which examines a different factor related to or affecting OSS development, adoption, and use. The topic of each major section, in turn, is examined by 7-10 authors from different cultures and backgrounds—authors who provide a broad range of perspectives on the related topic. As a result, each major section provides readers with both a more holistic treatment of each subject and a broad base of information upon which more informed decisions can be made.

The seven major sections of this handbook are as follows:

- **Section I: Culture, Society, and Open Source Software:** The entries in this section overview the internal culture of the individuals who create OSS products as well as examine both social perspectives on OSS use ant the potential OSS has to change societies.
- **Section II: Development Models and Methods for Open Source Software Production:** These chapters explore different methods for creating OSS products and discuss the benefits and the limitations of such methods as well as consider approaches for maximizing the more successful elements of such methods.
- **Section III: Evaluating Open Source Software Products and Uses:** Authors in this section both present models for assessing the effective uses of various OSS products and provide opinions on what makes some OSS items successful while others are not.
- **Section IV: Laws and Licensing Practices Affecting Open Source Software Uses:** In this section, chapters examine how legal factors and licensing strategies try to shape OSS development and use and also explore the new legal situations created by OSS products.
- **Section V: Public Policy, the Public Sector, and Government Perspectives on Open Source Software:** The chapters provide both examples of how government agencies and other non-profit organizations have adopted or adapted OSS use to meet programming needs; they also present ideas for how such public-sector entities should view OSS within the context of their activities.
- **Section VI: Business Approaches and Applications Involving Open Source Software:** This section's authors present models and cases for OSS development approaches and uses in for-profit endeavors as well as explore how business can address some of the more problematic aspects of OSS adoption and use.
- **Section VII: Educational Perspectives and Practices Related to Open Source Software:** Entries in this concluding section employ a range of perspectives and approaches to examine how OSS products can be integrated into educational activities in different contexts within and across societies.

While this collection of chapters provides readers with a wealth of OSS-related information, this text only begins to explore the complex environment in which software is operated. The foundation provided by the essays in this handbook, however, is an essential one for helping readers understand key concepts and ask the right questions when exploring software adoption and use. By using this information and building upon these ideas and perspectives, readers can enhance their views of software use in society while also shaping policies and practices related to software.

Kirk St.Amant and Brian Still
Lubbock, TX, USA
April 2007