Understanding Open Source Software Evolution

Walt Scacchi
Institute for Software Research
University of California, Irvine
Irvine, CA USA 92697-3425
wscacchi@uci.edu

October 2004 (Original version April 2003)
Revised version to appear in N.H. Madhavji, M.M. Lehman, J.F. Ramil and D. Perry (eds.), *Software Evolution and Feedback*, John Wiley and Sons Inc, New York, 2006

1. Introduction

This chapter examines the evolution of open source software and how their evolutionary patterns compare to prior studies of software evolution of proprietary (or closed source) software. Free or open source software (F/OSS) development focuses attention to systems like the GNU/Linux operating system, Apache Web server, and Mozilla Web browser, though there are now thousands of F/OSS projects underway. As these systems are being ever more widely used, questions regarding their evolution are of considerable interest.

This chapter is organized around four themes. First, it presents a brief survey of empirical studies of software evolution. As the majority of published studies of this kind are associated with the development of the laws of software evolution due to Lehman and colleagues, the kinds of findings they provide are described. Additionally, a sample of other empirical studies of software evolution are provided as well, in order to round out what is presently known about software evolution, at least in terms of studies of closed source software systems developed within centralized software development centers.

Second, it presents selected data and evidence that has begun to appear that characterizes change and evolution patterns associated with the evolution of F/OSS. Along the way, attention shifts to an analysis of where, how, and why the evolution of F/OSS does or does not conform to prior empirical studies, models, or theories of software evolution. Without revealing too much at this point, it is fair to say that there are patterns of data from studies of F/OSS that are not fully explained by prior studies of software evolution, as presently stated.

Third, it presents a brief review of models and theories of evolution from domains outside of software. This will help facilitate understanding of some of the challenges and alternative historical groundings that might be used to shape our collective understanding of how to think more broadly about software evolution, as well as the significance of theorizing about it.

The fourth and last section addresses whether it is necessary to reconsider the models, laws, and theory and how they can be modified and supplemented to better account for the observations and findings emerging in studies of new software development processes and environments, such as those associated with the development of F/OSS. Prior models of software evolution were developed based on careful study of conventional, closed source software systems that evolve within industrial settings.

Studies of F/OSS examine systems that typically are evolved outside of industrial settings, though some F/OSS systems are used in industrial settings, even though they are being developed and evolved now in both industrial or non-industrial environments. However, it is appropriate to consider how to update and revise the models, laws, and theory of software evolution to better account for both open and closed-source software system are being evolved inside or outside of industrial settings.

As such, the remainder of this chapter progresses through each of these themes in the order presented here.

2. Empirical Studies of Software Evolution

To understand the state of the art in the development of a theory of software evolution, and whether and how it might be extended, it is necessary to identify and describe what empirical studies of software evolution have been reported. However, it is encouraging to see that the empirical study of F/OSS evolution has recently become a topic of interest, and study has been recently taken up by several groups around the world.

2.1 Studies of the Laws of Software Evolution

The most prominent studies of software evolution have been directed by M.M. Lehman and colleagues over a 30 year period dating back to the mid-1970's. The studies have given rise to eight laws of software evolution, as formulated and refined by Lehman and colleagues [Lehman, *et al*., 1980-2001]. These laws are the result of careful and challenging empirical studies of the evolution of large-scale software systems found in a variety of corporate-based settings. These laws seek to consistently account for observed phenomena regarding the evolution of software releases, systems, and E-Type applications, as defined by Lehman and colleagues. The laws and theory can be reformulated in a manner suitable for independent test and validation, or refutation [Lakatos 1976, Popper 1961], but this requires making assumptions about details that are not explicitly stated in the laws, as explained in [Ramil 2002]. Thus there are many challenges in how such empirical testing of these laws should be performed (e.g., how many or what kinds of software systems constitute an adequate or theoretically motivated sample space for comparative study), what the consequences for refutation may be (rejection or reformulation of the laws/theory), and whether or how the laws and theory might be refined and improved if new or contradictory phenomena appear [cf. Glaser and Strauss 1976, Yin 1994].

The most recently published studies by Lehman and colleagues provide data from evolution of releases primarily from five software systems: two operating systems (IBM OS 360, ICL VME Kernel), one financial system (Logica FW), two versions of a large real-time telecommunications system, and one defense system (Matra BAE Dynamics). Other studies have also been conducted and found to yield consistent growth models, but their results are not widely available. The data is summarized as a set of growth curves, as described in Lehman, Ramil and Sandler [2001]. In plotting these growth curves as graphs, the X-axis denotes the sequence number of the software release that was analyzed, while the Y-axis denotes the size of the system (e.g., measured in the number of modules) after the first release. The graphs suggest that during its evolution (or maintenance process), a system tracks a growth curve that can be approximated either as

linear or inverse-square model [Turski 1996] within phases of the lifetime of an application, separated by transitions. The combined patterns can be interpreted as "S" curves, with phases of growth in size follow by less rapid growth (even stagnation) and transitions to another growth phase. Thus, these data/curves explicate conformity to the first (continual change), second (increasing complexity) and sixth (continual growth) laws, in that they suggest continual adaptation via incremental growth, system complexity controls the growth rate in a constant/bounded (linear or inverse-square) manner. The third, fourth and fifth laws have been refined to account for the new observations. The seventh law addressing quality could not be directly observed within the data, but may conform to observations made by Lehman and colleagues about these systems. The eighth law (feedback system) is a synthesis of the other laws. The support to each of the other laws strengthens the eight. In general, in the most recent studies by Lehman and colleagues their data set and the diversity of data substantiates and supports the original or refined versions of the laws [Lehman & Ramil 2002].

However, it is unclear whether such a data set is a representative sample of different kinds/types of software systems, or whether the laws can be interpreted as providing theoretical guidance for what kinds/types of software systems to study. It may be apparent that the majority of systems are medium to large or very large software systems developed and maintained in large corporate settings, that the customers for such systems are also likely to be large enterprises (i.e., they are not intended as software for a personal or hand-held computer). In addition, some of the software systems or associated data that were examined in the studies by Lehman and colleagues are confidential, and thus are not open for public inspection, or independent examination and assessment. Subsequently, students and other scholars cannot readily access these systems or data for further study.[1]

2.2 Other Empirical Studies of Software Evolution

Many other empirical studies have been conducted and published. Here attention is directed to a sample of these studies where non-open source software systems were being investigated. This is mainly intended to see if other studies of software evolution conform to, refute, or otherwise extend and refine the laws and theory of software evolution.

Bendifallah and Scacchi [1987] present qualitative data and analysis of two comparative case studies revealing that similar kinds of software systems in similar kinds of organizational settings have different evolutionary trajectories. They report that the differences can be explained by how system maintainers and end-users deal with local contingencies in their workplace and career opportunities in the course of maintaining their software systems.

Tamai and Torimitsu [1992] present data and observations from a survey study of mainframe software system applications across product generations. Among other things, they report that software lifetime in their survey is on average about 10 years, the variance in application lifetime is 6.2, and that small software applications tend to have a shorter live on average. They also report that applications that constitute what they call

[1] Early data from Lehman's studies can be found in one of his books [Lehman and Belady 1985]. Unfortunately, the *unavailability* of empirical data from software measurements studies is in general all too common of an occurrence. However, studies of F/OSS may indicate a different future lies ahead regarding public data availability [cf. Koch and Schneider 2000, Robles-Martinez, Gonzalez-Barahona, *et al*., 2003].

administration systems (e.g, back office applications) live longer than *business supporting* (i.e., mission-critical) systems, and that application systems that replace previous generation systems grow by more than a factor of 2 compared to their predecessors. Last, they report that some companies follow policies that set the predicted lifetime of an application system at the time of initial release, and use this information in scheduling migration to next generation systems.

Cusumano and Yoffie [1999] present results from case studies at Microsoft and Netscape indicating strong reliance on incremental release of alpha and beta versions to customers as business strategy for improving evolution of system features that meet evolving user requirements. They show that user satisfaction can improve and be driven by shortening the time interval between releases. They also find that unstable releases (e.g., alpha and beta versions) will be released to users as a way to enable them to participate in the decentralized testing and remote quality assurance, and thus affecting software evolution. Their study does not confirm or refute the laws of software evolution, but they introduce a new dynamic into software evolution by making the release activity an independent output variable rather than an input variable.

Gall, Jayazeri, *et al*., [1997] provide data and observations based on software product release histories from a study of a large telecommunications switching system. The growth of this system over twenty releases conforms to the general trends found in the data of Lehman and colleagues. However, they report that though global system evolution follows the trend and thus conforms to the laws, individual subsystems and modules do not. Instead, they sometimes exhibit significant upward or downward fluctuation in their size across almost all releases. Eick, Graves, *et al.*, [2001] also provide data demonstrating that source code decays unless effort and resources are allocated to prevent and maintain the system throughout the later stages of its deployment, and that the decay can be observed to rise and fall in different subsystems and modules across releases.

Kemerer and Slaughter [1999] provide a systematic set of data, analyses, and comparison with prior studies revealing that problems in software maintenance can be attributed to a lack of knowledge of the maintenance process, and of the cause and effect relationships between software maintenance practices and outcomes. However, they do observe that their data may be associated with the growth of system complexity and other outcomes over time, which they attribute to the laws observed by Lehman and colleagues.

Perry, Siy, and Votta [2001] report findings from an observational case study of the development of large telecommunications systems that indicates extensive parallel changes being made between software system releases. This notion of parallel changes that may interact and thus confound software maintenance activities is not accounted for in an explicit way by the laws of software evolution. Thus, it does introduce yet another organizational factor that may affect software evolution.

With the exception of Cusumano and Yoffie [1999], these studies either conform to or suggest extensions to the laws and theory of software evolution. Thus these conditions may point to the need for either revisions to the laws, or alternative theories of software evolution that may or may not depend on such laws.

3. Evolutionary Patterns in Open Source Software

F/OSS development has appeared and disseminated throughout the world of software technology, mostly in the last ten years. This coincides with the spread, adoption, and routine use of the Internet and World Wide Web as a global technical system. This infrastructure supports widespread access to previously remote information and software assets, as well as the ability for decentralized communities of like-minded people to find and communicate with one another. This is a world that differs in many ways from that traditional to software engineering, where it is common to assume centralized software development locales, development work, and administrative authority that controls and manages the resources and schedules for software development and maintenance. Thus to better understand whether or how patterns of software evolution in the technical and social regime of F/OSS conform to or differ from prior studies or models of software evolution, then it is appropriate to start with an identification of the types of entities for F/OSS evolution, then follow with an examination of empirical studies, data and analyses of F/OSS evolution patterns.

3.1 Types of entities for studying F/OSS evolution

The scheme of objects types that are suitable to address in studies of software evolution have been identified in the studies by Lehman and colleagues over the years [cf. Lehman 1980, 2002]. The primary types of entities are software releases, systems, applications, development processes, and process models. Accordingly, each of these can be cast in terms of F/OSS as follows.

F/OSS Releases — In general, large F/OSS systems continue to grow over time and across releases. This suggests consistency with the sixth law of software evolution. Both stable and unstable F/OSS release product versions are being globally distributed in practice. Periodic alpha, beta, candidate, and stable releases are made available to users at their discretion, as are unstable nightly F/OSS build versions released for developers actively contributing software updates to a given release. F/OSS releases for multiple platforms are generally synchronized and distributed at the same time, though may vary when new platforms are added (in parallel). F/OSS releases thus evolve within a non-traditional process cycle between full stable releases. F/OSS releases are also named with hierarchical release numbering schemes, sometimes with three or four levels of nested numbering to connote stable versus unstable releases to different audiences. However, the vast majority of F/OSS systems, primarily those for small and medium size F/OSS systems, do not continue to grow or thrive, perhaps because the software is not intensively or widely used [Capiluppi, Lago, and Morisio 2003].

F/OSS Systems – F/OSS systems or programs evolve from first statement of an application concept or a change required, to an existing system released and installed as an operational program text with its documentation. F/OSS systems may be small (<5K SLOC[2]), medium (5K-100K SLOC), large (100K-1000K SLOC) or very large systems (>1MSLOC), with large and very large systems being the fewest in number, but the most widely known. Most large or very large F/OSS systems or programs may exist in related but distinct versions/releases intended for different application platforms (e.g., MS

[2] Source Lines of Code, where 50 SLOC represents the equivalent of one printed page of source code, single spaced.

Windows, Solaris, GNU/Linux, Mac OS X). Many F/OSS are structured as distributed systems, systems configured using scripts (e.g., using Perl, Python, Tcl), middleware, or as modules that plug-in to hosts/servers (e.g., Apache, Mozilla, and now Firefox support independently developed plug-in modules). Additionally, some F/OSS are dynamically linked systems configured at run-time, when developed in a programming language like Java or others enabling remote service/method invocation.

F/OSS Applications – A much greater diversity and population of F/OSS applications are being investigated for evolution patterns. Those examined in-depth so far include the Linux Kernel, Debian Linux distributions[3], Mono, Apache Web server, Mozilla Web browser, Berkeley DB, GNOME user interface desktop, PostgreSQL DBMS, and about a dozen others[4]. Studies of F/OSS application populations, taxonomy, and population demographics for hundreds to upwards of 40K F/OSS systems have appeared [Madey, Freeh, and Tynan 2002].

F/OSS Process – F/OSS are developed, deployed, and maintained according to some software process. It is however unclear whether F/OSS processes, as portrayed in popular literature [DiBona, Ockman and Stone 1999], are intended only to be viewed as a monolithic process, just the top-level of a decomposable process, or whether specific software engineering activities have distinct processes which may also evolve, either independently or jointly. Furthermore, a small number of recent studies have begun to observe, describe and compare F/OSS development processes with those traditional to software engineering [Reis and Fortes 2002, Mockus, Fielding, and Herbsleb 2002, Scacchi 2002a,b, 2004] that point to differences in the activities and organization of the F/OSS process. In addition, F/OSS activities surrounding software releases may have their own distinct process [Erenkrantz 2003, Jensen and Scacchi 2003] that may not reflect the activities involved in the release of closed-source systems examined in the preceding section.

Models of F/OSS Process – Existing models of software development processes [Scacchi 2002b] do not explicitly account for F/OSS development activities or work practices [cf. Scacchi 2002a, 2002c, Jensen and Scacchi 2003]. Thus it is unclear whether models of software evolution processes that characterize closed-source software systems developed within a centralized administrative authority can account the decentralized, community-oriented evolution of F/OSS.

Overall, evolving software systems may be packaged and released in either open source or closed source forms. The packaging and release processes and technical system infrastructure may at times differ or be the same, depending on the software system application and development host (e.g., a Web site for open source, a corporate portal for closed source). But the decentralized community-oriented technological regime and

[3] A GNU/Linux distribution includes not only the Kernel, but also hundreds/thousands of utilities and end-user applications. Distributions are typically the unit of installation when one acquires GNU/Linux, while the Linux Kernel is considered the core of the distribution. However, many F/OSS applications are developed for non-Linux Kernel operating systems (e.g., MS Windows), thus assuming little/no coupling to the Linux Kernel.

[4] Smith, Capiluppi, and Ramil [2004] have published preliminary results from an ongoing comparative study of 26 OSS systems and applications. Such studies begin to suggest that future studies of software evolution will focus attention to F/OSS for a variety of reasons.

infrastructure of F/OSS appears different than the world of the centralized corporate-centered regime and infrastructure of the closed source systems that have been examined as the basis of the laws of software evolution. Nonetheless, the laws of software evolution seem to apply, at least at a very high level, in accounting for the evolution of F/OSS.

3.2 Patterns in Open Source Software Evolution Studies

Attention is now directed to examples of studies where F/OSS systems are being investigated, with the focus on how their results can be compared with those of Lehman and colleagues.

Godfrey and Tu [2000] provide data on the size and growth of the Linux Kernel (2M+ SLOC) from 1994-1999, and find the growth rate to be super-linear (i.e., greater than linear), as portrayed in Figures 1 through 3. They also find similar patterns in F/OSS for the Vim text editor. Schach, Jin, *et al.*, [2002] report on the result of an in-depth study of the evolution of the Linux Kernel across 96 releases [cf. Godfrey and Tu 2000] indicating that module coupling (or interconnection) has been growing at an exponential (super-linear) rate. Their data are displayed in Figure 4. They predict that unless effort to alter this situation is undertaken, the Linux Kernel will become unmaintainable over time. Koch and Schneider [2000] studied the GNOME user interface desktop (2M+ SLOC) and provided data that shows growth in the size of the source code base across releases increases in a super-linear manner as the number of software developers contributing code to the GNOME code base grows. Data from their study is plotted in Figure 5.

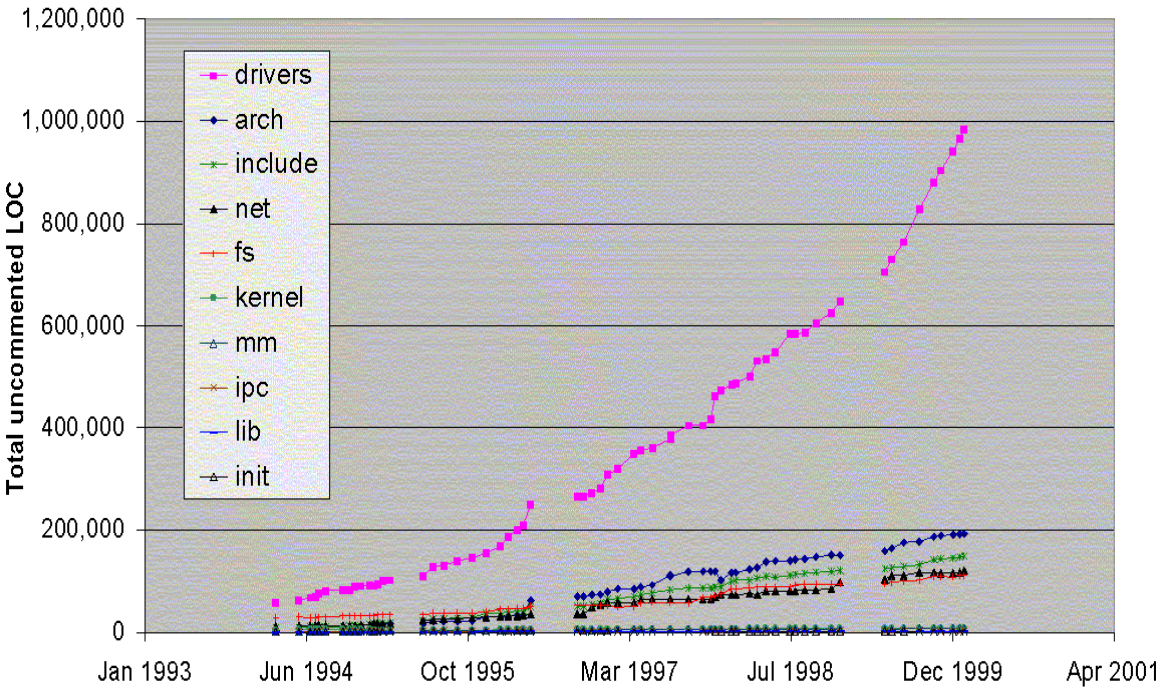

Figure 1. Data revealing the size and growth of major sub-systems in the Linux Kernel during 1994-1999 [Source: Godfrey and Tu 2000].

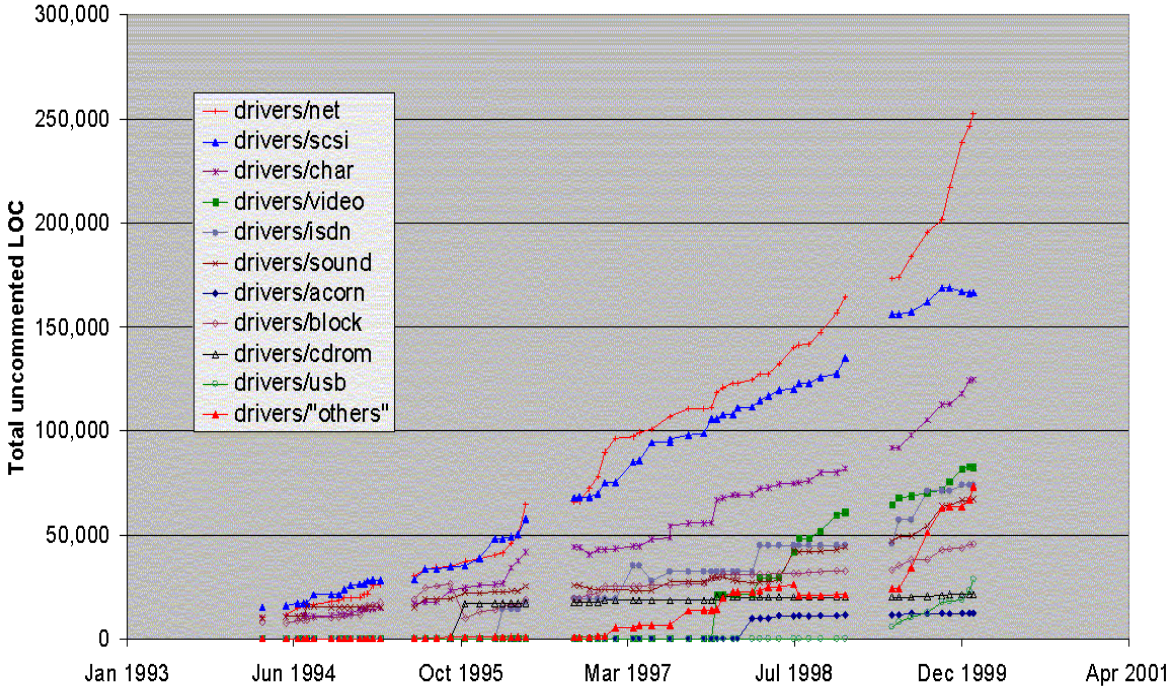

Figure 2. Data revealing the size and growth of device drivers in the Linux Kernel during 1994-1999 [Source: Godfrey and Tu 2000].

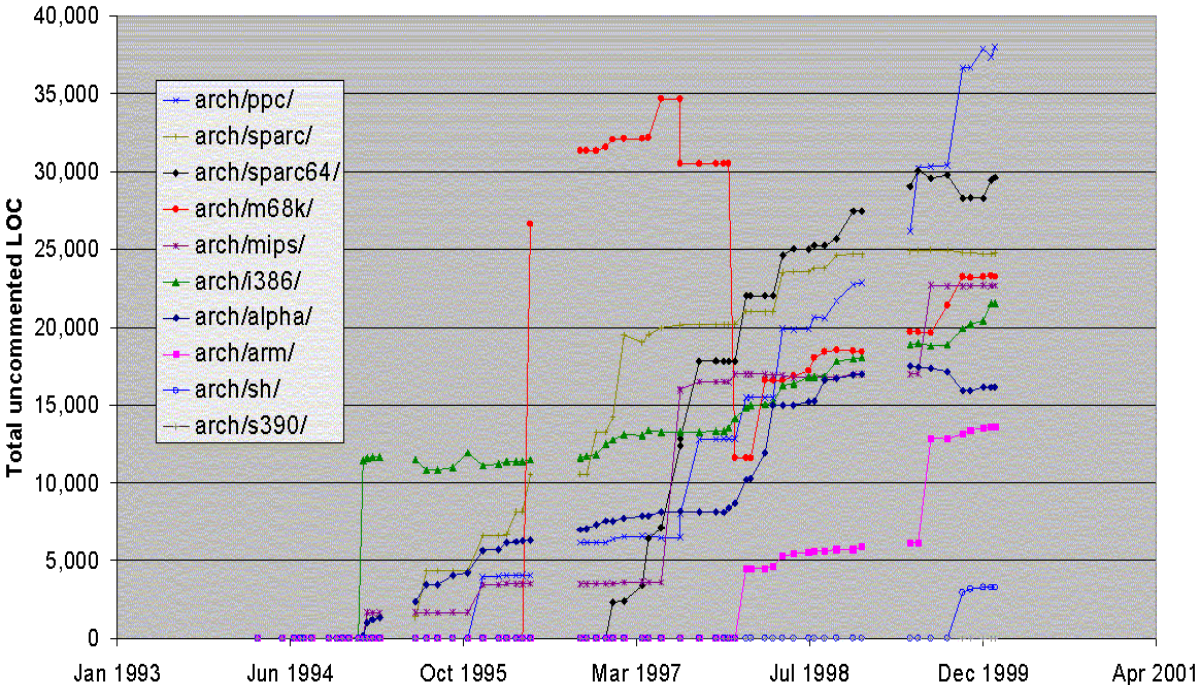

Figure 3. Data revealing the size and growth of the Linux Kernel for different computer platform architectures during 1994-1999 [Source: Godfrey and Tu 2000].

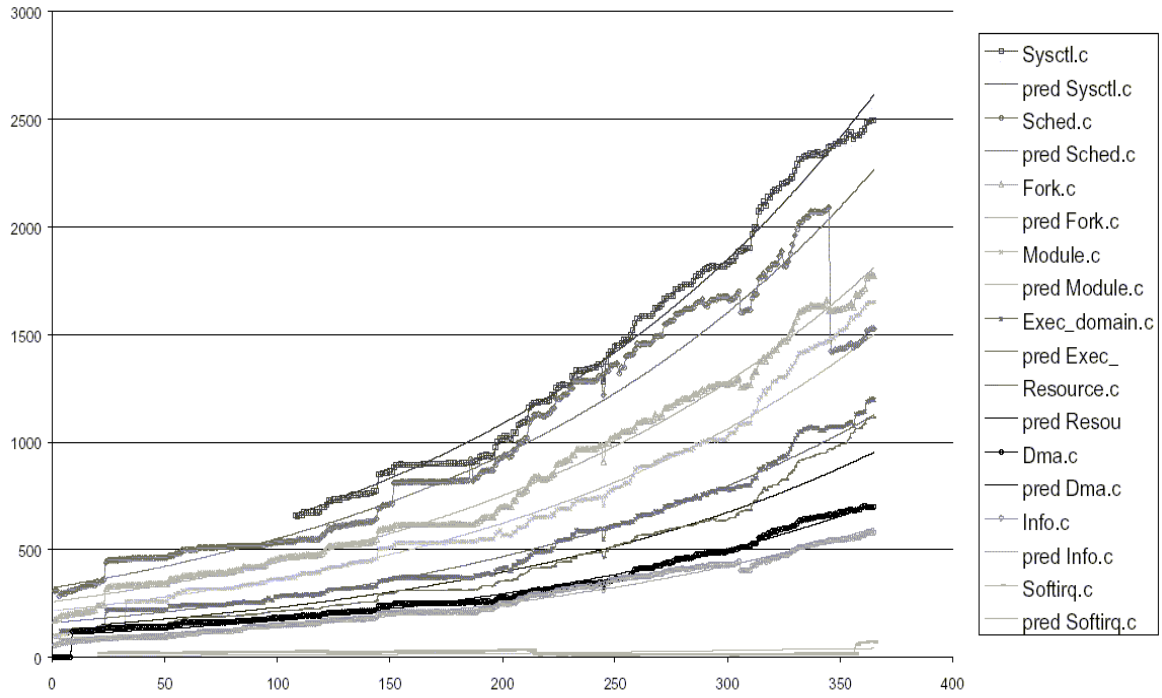

Figure 4. Measured (discrete points) versus predicted (smooth curves) of common coupling of source code modules in the Linux Kernel across releases [Source: Schach, Jin, *et al*., 2002].

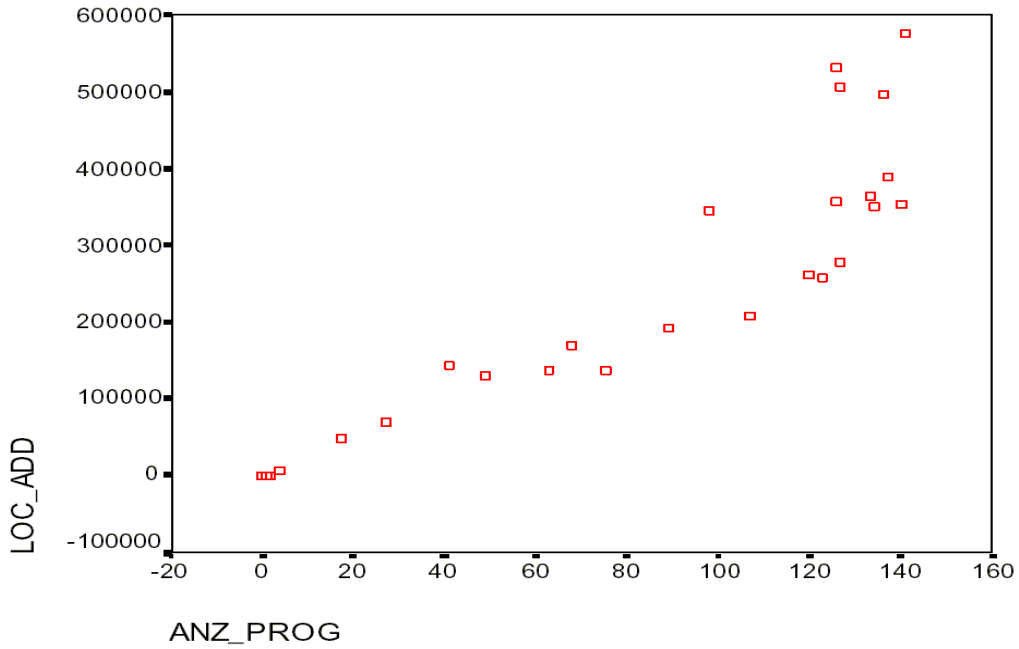

Figure 5. Growth of the lines of source code added as the number of software developers contributing code to the GNOME user interface grows [Source: Koch and Schneider 2000].

Robles-Martinez, Gonzalez-Barahona, *et al.*, [2003] report in their study of Mono (a F/OSS implementation of Microsoft's .NET services, libraries, and interfaces), their measurements indicate super-linear growth rate in the code size and the number of code updates that are committed within the code base. They also report a similar growth pattern in the number of people contributing source code to the emerging Mono system over a 2-3 year period. According to Gonzalez-Barahona, Ortuno Perez, *et al*, [2001] their measurements indicate that as of mid-2001, the Debian GNU/Linux 2.2 distribution had grown to more than 55M SLOC, and has since exceeded 100M SLOC in the Debian 3.0 distribution. O'Mahony [2003] presents data from her study of the Debian Gnu/Linux distribution from releases spanning 0.01 in 1993 through 3.0 in late 2002 that show growth of the size of the distribution rises at a super-linear rate over the past five years. Last, Gonzalez-Barahona, Lopez, and Robles [2004] also provide data on the growth of the Apache project community and number of modules, revealing once again, a super-linear growth pattern over the five year period (1999-2004) covered in their data.

In contrast, Godfrey and Tu [2000] find linear growth in Fetchmail, X-Windows, and Gcc (the GNU compiler collection), and sub-linear growth in Pine (email client). Such trends are clearly different from the previous set of F/OSS systems.

Why is there such a high growth rate for some F/OSS systems like the Linux Kernel, Vim, GNOME, Mono, the Debian GNU/Linux distribution, and the Apache project, but not for other F/OSS? Godfrey and Tu [2000] report in the case of the Linux Kernel that (a) much of the source code relates to device drivers, as seen in Figure 2, (b) much of the code is orthogonal and intended for different platforms, as suggested in Figure 3, and (c) contributions to the code base are open to anyone who makes the requisite effort. In addition, Godfrey and Tu observe (d) Linux Kernel source code configurations (or "builds") are specific to a hardware platform or architecture (see Figure 3), and use as little of 15% of the total Linux Kernel source code base. It is possible but uncertain whether these conditions also apply to GNOME, Vim, Mono and the Apache project, since they may have source code configurations that are specific to different operating systems (Linux, BSD, Windows, or Mac OS/X). However, it is unclear why they would or would not apply to Fetchmail, X-Windows, Gcc and Pine. Perhaps it might be because the latter systems are generally older and may have originally been developed in an earlier (pre-Web) technological regime. Elsewhere, Cook, Ji, and Harrison [2000] in their comparison study of the closed-source Logica FW system, and the F/OSS Berkeley DB system, find that growth across releases is not uniformly distributed, but concentrated in different system modules across releases. A similar result may be seen in the data in Figure 3, from Godfrey and Tu [2000].

Nakakoji, Yamamoto, *et al*., [2002] report findings from a comparative case study of four F/OSS systems, the Linux Kernel, Postgres DBMS, GNUWingnut, and Jun a 3D graphics library. They provide data indicating that these systems exhibit different evolutionary patterns of splitting and merging their overall system architectures across releases, as shown in Figure 6. Thus it appears that it is necessary to understand both the *age* and *architectural patterns* of sub-systems and modules within and across software releases, whether in closed source or open source systems, in order to better understand how a system is evolving [Godfrey and Lee 2000]. This observation is also implicated by earlier

studies [Tamai and Torimitsu 1992, Gall, Jayazeri, *et al*. 1997, Eick, Graves, *et al.* 2001, Perry, Siy, and Votta 2001].

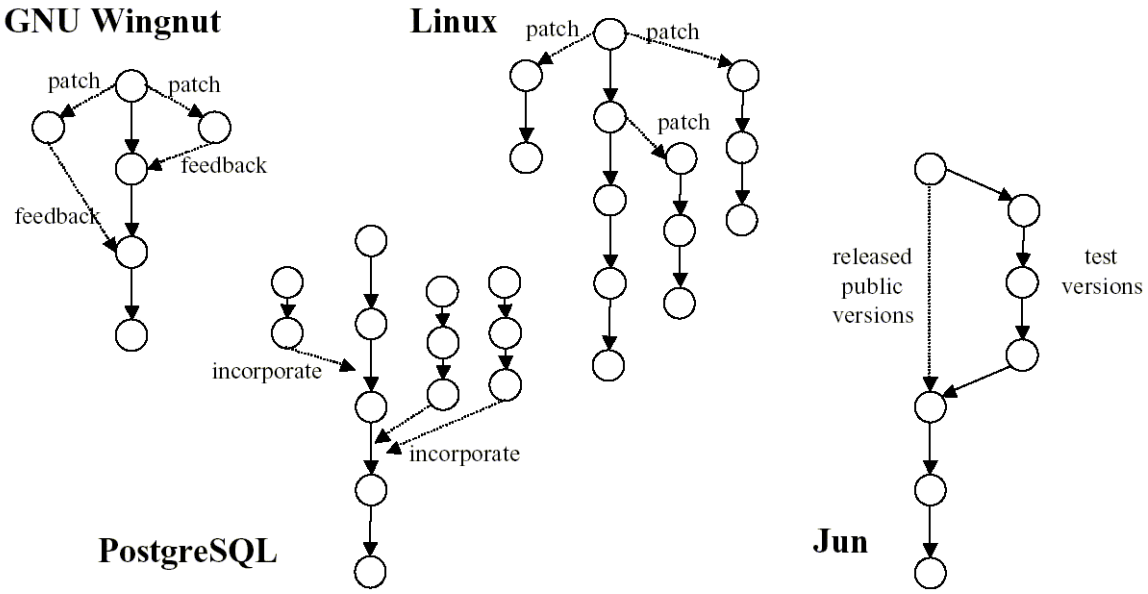

Figure 6. Patterns of software system evolution forking and joining across releases (nodes in each graph) for four different F/OSS systems
[Source: Nakakoji, Yamamoto, *et al*., 2002]

Hunt and Johnson [2002] report discovery of a Pareto distribution in the size of the number of developers participating in F/OSS projects, from a sample population of >30K projects found on the SourceForge Web portal.[5] Their results indicate that the vast majority of F/OSS projects have only one developer, while a small percentage have larger, ongoing team membership. Madey, Freeh, and Tynan [2002] in an independent study similar to Hunt and Johnson, find that a power law distribution characterizes the size of F/OSSD projects across a population of some 40K F/OSS projects at SourceForge. Independently, Hars and Ou [2002] report a similar trend, finding that more than 60 percent of F/OSS developers in their survey reported participating in 2-10 other F/OSS development projects. Capilupppi, Lago, and Morisio [2003] also draw from a sample of 400 F/OSSD projects posted on SourceForge. They find that the vast majority of systems in their sample are either small or medium size systems, and only a minor fraction are large. Only the large F/OSS systems tend to have development teams with more than a single developer. Their results might also be compared to those of Tamai and Torimitsu [1992], thereby substantiating that small F/OSS systems have a much shorter life, compared to large F/OSS systems. Overall, this suggests that results from studies that characterize large F/OSS efforts are not representative of the majority of F/OSS projects.

[5] The SourceForge Web portal can be found at www.sourceforge.net. At the moment, there are 85K F/OSS projects now registered at this specific F/OSS project portal. Other F/OSS Web portals like www.freshmeat.org and www.savannah.org include other projects, though there is some overlap across these three portals.

Di Penta, Neteler, *et al.*, [2002] provide results from a case study focused on the refactoring of a large F/OSS application, a geographical information system called GRASS, which operates on a small hand-held computer. Their effort was aimed at software miniaturization, reducing code duplications, eliminating unused files, and restructuring system libraries and reorganizing them into shared (i.e., dynamically linked) libraries. This form of software evolution and architectural refactoring has not been reported in, or accounted for by, the laws of software evolution. For example, miniaturization and refactoring will reduce the size of the software application, as well as potentially reducing redundancies and code decay, thereby improving software quality. Elsewhere, Scacchi [2002c] reports results from a case study of the GNUenterprise project that find that the emerging F/OSS E-Commerce application system being developed is growing through merger with other independently developed F/OSS systems, none of which was designed or envisioned as a target for merger or component sub-system. He labels this discontinuous growth of F/OSS system size and functionality, *architectural bricolage.* Such capabilities may account for the discontinuities that can be seen in the growth trends displayed in Figure 3.

Mockus, Fielding, Herbsleb [2002] in a comparative case study of Apache Web server (<100K SLOC) and Mozilla Web browser (2M+ SLOC), find that it appears easier to maintain the quality of system features for a F/OSS across releases compared to closed-source commercial telecommunications systems of similar proportions. They also find evidence suggesting large F/OSS development projects must attain a core developer team size of 10-15 developers for its evolution to be sustained. This might thus be recognized as an indicator for *a critical mass in the number of core developers* that once achieved enables a high rate of growth and sustained viability. Whether and how long such growth can be sustained however is unclear as the number of core developers changes over time.

Scacchi and colleagues [2002a, 2002c, Elliott and Scacchi 2002, Jensen and Scacchi 2003] provide results from comparative case studies of F/OSS projects within different communities. They find and explicitly model how F/OSS requirements and release processes differ from those expected in conventional software engineering practices. They also find that evolving F/OSS depends on co-evolution of developer community, community support software, and software informalisms as documentation and communication media. Nakakoji, Yamamoto, *et al*., [2002] also report that the four F/OSS systems they investigated co-evolve with the communities of developers who maintain them. Finally, Gonzalez-Barahona, Lopez, and Robles [2004] provide a detailed data set that visualizes the growth of the developer community over a five year period corresponding to growth in the number of modules incorporated in the Apache project.

Von Hippel and Katz [2002] report results of studies that reveal some end-users in F/OSS projects become developers, and most F/OSS developers are end-users of the systems they develop, thereby enabling the co-evolution of the system and user-developer community. This observation of developers as users as developers is also independently reported in other studies as well [Mockus, Fielding, Herbsleb 2002, Scacchi 2002a, and Nakakoji, Yamamoto, *et al*., 2002]. Last, much like Hars and Ou [2002], Madey, Freeh, and Tynan [2002] report finding that some F/OSS developers, whom they designate as *linchpin developers*, participate in multiple projects. These linchpin developers effectively create social networks that interlink F/OSS projects and enable the systems

interlinked in these social networks to also share source code or sub-systems. A sample from their data appears in Figure 7.

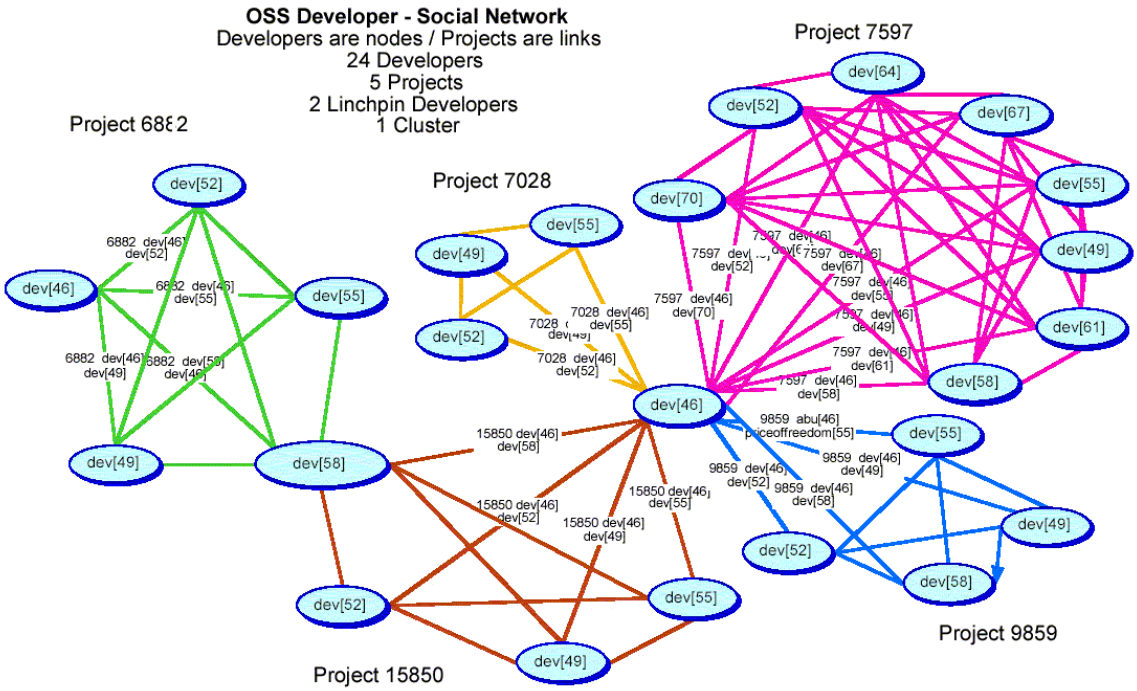

Figure 7. A social network of F/OSS developers that interlinks five different projects through two linchpin developers, dev[46] and dev[58]
[Source: Madey, Freeh, and Tynan 2002].

However, across the set of studies starting above with Mockus, Fielding and Herbsleb [2002], there are no equivalent observations or laws reported in prior studies of closed source software evolution that account for these data and evolutionary patterns addressing team and community structures. Clearly, such a result does not imply that the observed conditions or patterns do not occur in the evolution of closed source software. Instead, it reveals that other variables and patterns previously not addressed in prior empirical studies may be significant factors contributing to software evolution.

Last, in a recent study comparing open versus closed source software development products, Paulson, Succi, and Eberlein [2004] find that overall evolutionary growth of both types of software are comparable, and consistent with the laws of software evolution, for the systems they examined. Specifically, in modeling and visualizing the growth of the systems in their studies, as displayed in Figure 8, their research design employs linear approximations to depict system growth trends over time. Thus, it is not possible to tell from their results if these approximations "linearize" the inverse-square growth curves reported by Lehman and colleagues or the exponential curves of the kind shown in Figures 1 through 5, or the non-linear growth shown in Figure 6. However, they do plot a growth curve for the Linux as seen in Figure 8, which may suggest their linear approximation in this instance flattens the exponential growth pattern for Linux seen in Figures 1 through 5.

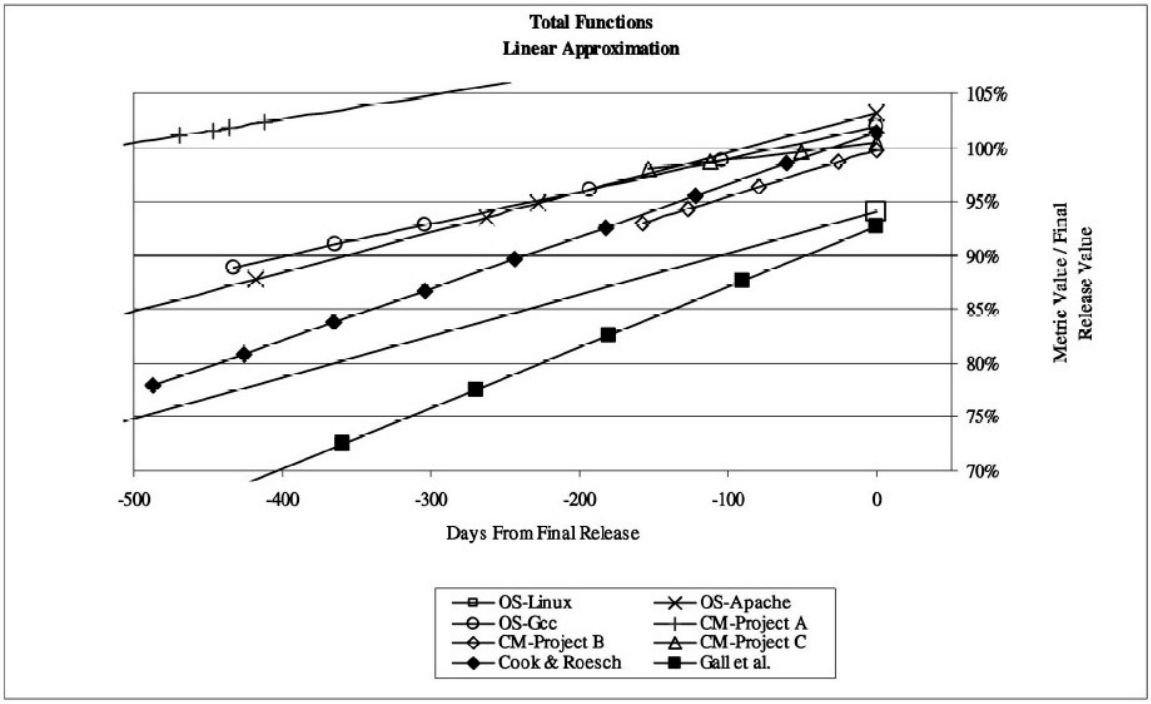

Figure 8. Linear approximations of the growth of a sample of open and closed source software systems [Source: Paulson, Succi, and Eberlein, 2004]

Overall, in evolving large F/OSS, it seems that it may be necessary for a critical mass of developers to come together to anchor the broader community of users, developers, and user-developers to their shared system. This critical mass and community will co-evolve with the architectural patterns that are manifest across unstable and stable F/OSS system releases as they age over time. Subsequently, older F/OSS systems that may have emerged before the F/OSS gained widespread recognition as a social movement and cultural meme, may have a lower rate of architectural and system release co-evolution. Furthermore, it may well be the situation that for large F/OSS systems/releases to evolve at a super-linear rate, that this may be possible only when their development community has critical mass, is open to ongoing growth, and that the focal F/OSS systems entail internal architectures with orthogonal features, sub-systems, or modules, as well as external system release architectures that span multiple deployment platforms.

Last, it appears that the evolutionary patterns of F/OSS systems reveal that overall system size and architecture can increase or decrease in a dis-continuous manner, due to bricolage-style system mergers, or to miniaturization and refactoring. Clearly, the laws of software evolution as presently stated, and based primarily on the study of large closed source systems, do not account for, nor anticipate, the potential for super-linear growth in software system size that can be sustained in the presence of satisfied developer-user communities who collectively assure the quality of these systems over time.

4. Evolution Models and Theories

As a first step, it is desirable to provide a definition of evolution that is satisfied by examples that cover different scientific and technological disciplines. Lehman and Ramil [2004] provide an appropriate definition for software evolution. In the definition that follows, an attempt has been made to address properties applicable in a general sense. Individual disciplines may have additional properties not identified here. Accordingly, evolution is a process of progressive change and cyclic adaptation over time in terms of the attributes, behavioral properties, and relational configuration of some material, abstract, natural or artificial entity or system. Such a definition accommodates both "evolutionistic" models that draw attention to stages and direction of developmental progress, and "evolutionary" models that focus attention to mechanisms, forces, or impinging constraints that give rise to evolution [cf. King and Kraemer 1984].

Theories of biological evolution have been the subject of scientific inquiry and speculation for centuries, with Charles Darwin's *Origin of Species* (1859) being the most widely known and cited theory. Darwin's theoretical analysis was based in part on his field studies of animal species on the Galapagos archipelago. His theory begat a century of scientific debate that included religious and moral substance undertones [Bowler 1989]. It led to the emergence of concepts such as *developmental biology* that examines the role of genetics, reproductive (natural) selection, co-evolution (among co-located species) and adaptation to ecological circumstances shaping the lives of organisms. In contrast, *evolutionary biology* accounts for the influence of genetics, reproduction, lineage, speciation, and population growth and diffusion shaping the long-term or trans-generational lives of species of organisms. The concepts of developmental versus evolutionary biology help draw attention to two alternative ways to view the evolution of a system, one focusing on a system's life cycle, the other on changes manifest across generations of related systems. In addition, the concept of biological *systematics* further helps draw attention to the "progress", direction, or (punctuated) equilibrium of evolution based on associating the developmental properties (e.g., agency, efficiency, and scope) of living organisms with those found in the fossil and geological record [Nitecki 1988, Gould 2002].

Culture, language and economy, which arise from the social actions and interactions of people, may evolve in ways similar or different from that in the natural science of biology. Culture, for example, may rely on the development, diffusion, and assimilation of *memes* (i.e., concepts, compelling ideas, or cultural "genes") that embody recurring social practices, situations, beliefs, or myths that can be shared, communicated, or otherwise transported as a basis for their evolution [Gabora 1997]. Despite differences, "open source" and "free software" are examples of related memes. Thus, rather than conjecturing physical (biological) conditions or circumstances, cultural evolution relies on social actions and narrative records that relate to physical conditions and circumstances that enable the ongoing evolution of diverse cultures and cultural experiences. Language evolution [Christiansen and Kirby 2003] seems to share and span ideas from culture and biology with respect to efforts that associate language learning, perception, and semiotics with neurological mechanisms and human population dynamics. Elsewhere, topics like *competition*, *resource scarcity*, *population concentration/density*, *legitimacy*, and *organizational ecologies* appear as factors shaping the evolution of markets, organizations and economies, at least at a macro level [Hannan

and Carroll 1992, Nelson and Winter 1982, Saviotti and Mani 1995]. Beyond this, the evolution of culture, language and economy are being explored experimentally using computational approaches [e.g., Gabora 2000]. Overall, this tiny sample of work draws attention to associations more closely aligned to evolutionary biology, rather than to developmental biology.

The evolution of modern technology has also become the subject of systematic inquiry. For example, in Abernathy's [1978] study of the American automobile industry, he finds that the *technical system* for developing and manufacturing automobiles associates product design and process design within a *productive unit* (i.e., the manufacturing systems within a physical factory or production organization). Each depends on the other, so that changes in one, such as the introduction of new techniques into a productive unit, are propagated into both product design and production process layout/workflow. King and Kraemer [1984] provide similar findings in their analysis of the evolution of computing systems in organizational settings. Hughes [1987] in his historical study of the technical system of electrification draws attention to the role of the *infrastructure* of electrical production and distribution as spanning not just equipment, mechanisms (e.g., power generators, sub-stations), cabling and power outlets, but also the *alignment* of producers, retailers, and consumers of devices/products together with the processes that depend on electrification for their operation. Meyer and Utterback [1994] were among the first to recognize that productive units and technical systems of production and consumption were increasingly organized around *product lines* that accommodate a diversity of product life cycles centered around the *dominant design* [Utterback 1994] or product architecture that dominates current retail markets.

From an economic perspective, Nelson and Winter [1982] independently termed the overall scheme that associates and aligns products, processes, productive units with producers, retailers and consumers, a *technological regime*. Last, though the development, use, and maintenance of software is strongly dependent on computer hardware, there are now studies that examine how different kinds of computer hardware components exhibit evolutionary patterns across technological generations or regimes [e.g., Victor and Ausubel 2002, van de Ende and Kemp 1999]. The evolution of technology through technological regimes that depend on product features, development processes, infrastructure, and productive units seems immediately relevant to understanding the evolution of software systems and technologies.

Software programs, systems, applications, processes, and productive units continue to develop over time. For example, there is a plethora of software innovations in the form of new tools, techniques, concepts or applications, which continue to emerge as more people experience modern computing technology and technical systems. These innovations give rise to unexpected or unanticipated forms of software development and maintenance, as, for example, software systems that are dynamically linked at run-time instead of compile-time [Mens *et al.*, 2003, Kniesel *et al.*, 2002]. Software innovations are diffused into a population of evermore diverse settings and technical systems via technology transfer and system migration. Software processes are subject to ongoing experience, learning, improvement and refinement, though there is debate about how to most effectively and efficiently realize and assimilate such process improvements [Conradi and Fuggetta 2002, Beecham, Hall and Rainer 2003]. Software systems are also subject to cultural

forces [Elliott and Scacchi 2002], narrative and informal documentation [Scacchi 2002a] and economic conditions [Boehm 1981] within the productive units or work settings that affect how these systems will be developed and maintained. These forces can give rise to similar kinds of systems in similar settings evolving at different rates along different trajectories [Bendifallah and Scacchi 1987]. This suggests that software systems are developed and evolved within particular organizational and informational ecologies [cf. Nardi and O'Day 1999], as well as situated within a technical system of production and larger overall technological regime.

Overall, this brief review of evolutionary theory across a sample of disciplines raises an awareness of the following issues. First, in studying software evolution, it is necessary to clarify whether in fact attention is directed at matters more closely aligned to development of a given system throughout its life, or with the evolution of software technologies across generations that are disseminated across multiple populations. It appears that much of what are labeled as studies of "software evolution" are more typically studies of patterns of development of specific systems, rather than patterns of evolution across different systems within one or multiple product lines (or species), at least as compared to work in biological evolution. However, the laws and theory of software evolution articulated by Lehman and associates depend on empirical findings that examine a variety of software systems in different application domains, execution environments, size of system, organization, and company marketing the system, as their basis for identifying mechanisms and conditions that affect software evolution.

Second, when considering the subject of software evolution at a macro level, it appears that there are no easily found or widely cited studies of that examine issues of memes, competition, resource scarcity, population concentration/density, legitimacy, and organizational ecology as forces that shape or impinge on software systems or software technology. The study of software evolution is still in its infancy. In general, existing theory of the development or evolution of software does not yet have a substantial cultural, language, or economic basis. Studies, analyses, and insights from these arenas are yet to appear, and thus need to be explored.

Last, conventional closed source software systems developed within centralized corporate productive units and open source software systems developed within globally decentralized settings without corporate locale represent alternative technological regimes. Each represents a different technical system of production, distribution/retailing, consumption, differentiated product lines, dominant product designs and more. Similarly, software development methods based on object-oriented design and coding, agile development, and extreme programming entail some form of alternative technological regime. Concepts from theories of technological evolution and observations on patterns of software development and maintenance can be used to help shape an understanding of how software evolves. Additional work is required to compare and contrast evolutionary behavior under different software development regimes. The present discussion concentrates on the socio-technical regime of open source software development.

5. Do We Need New or Revised Models, Laws, or Theories for Open Source Software Evolution?

At this point it is reasonable to ask about whether prior studies, models, or laws adequately account for the evolution F/OSS systems, at least according to the studies and data presented above. For example, other studies of software evolution do not provide a suitable account for the sometimes super-linear, sometimes sub-linear growth curves reported in studies and figures of F/OSS presented above. Beyond this, data trends and patterns accounting for the evolution of F/OSS in some cases conform, and in other cases it is unclear whether or how different F/OSS conform to the laws of software evolution. As such, refining or reformulating them to account for the data at hand is beyond the scope of this chapter. However, it is possible to consider the underlying ontologies for software evolution to rethink what kinds of theory or models of software evolution may further help in understanding, reasoning about, and explaining the evolution of both closed and open source software systems.

5.1 Embracing the Feedback Control Systems Ontology

Feedback and feedback systems appear to be a central part of the conceptual foundation of the laws and theory of software evolution developed by Lehman and colleagues. So why not refine these laws in a way that more fully embraces feedback control theory in articulating the laws so that they can address the evolution of F/OSS? The system dynamics modeling and simulation approach has been widely used to study software project management [Abdel-Hamid and Madnick 1991] and various types of software processes. However, the approach can also be used to model and simulate feedback control systems expressed as a system of differential equations [Bateson 1993, Dolye, Francis, and Tannenbaum 1992]. Combining the laws of software evolution with modeling concepts from system dynamics and feedback control systems should be possible with an eye toward the interests of the audience for whom the laws are intended to serve. For example, executives and senior managers responsible for large software development centers want to know where to make strategic investments and how to better allocate their software development staff, schedules, and related resources. Software developers may want to know what kinds of tools and techniques to use to make their software evolution efforts faster, better and cheaper. Scholars of software evolution theory want to know what kinds of data to collect, what types and sizes of systems to study, what types of criteria to use in designing theoretically motivated samples of software systems under study, and what tests to apply to verify, refine, or refute the laws or theory at hand. In terms of feedback control systems, there is need to identify where sensors should be placed in a software productive unit to collect different types of software change data, and to what or whom they should provide their feed back.

Similarly, there is need to identify where feedback control loops are to be placed, where their begin and end points are to be located, what functionality is located within each loop, and what decision function determines whether a loop iterates or exits. It is also necessary to identify what roles people and software tools play in the regulation or control of the feedback system, or what feedback they produce, use, or consume along the way. Managers, developers, and scholars want to know how different types of feedback get employed to regulate and control the centralized corporate or decentralized

open source productive unit that develops and maintains software systems of different size, type, age, and application setting.

Recent efforts by Lehman, Ramil and Kahen [2001], for example, have begun to employ system dynamics modeling techniques and simulation tools to demonstrate and iteratively refine an operational model of software evolution that embodies the existing laws. Their model seems able to reproduce via simulation the evolutionary data trends that conform to the laws of software evolution. More recently, Ramil and colleagues [Smith, Capiluppi, and Ramil 2004] examine and qualitatively simulate F/OSS evolution data for 26 systems they have studied. Their data and analyses from this latest study confirm and incorporate further refinements to the laws of software evolution, though they also find some puzzling trends that are not well explained by the laws. However, the trends reported in their data appear to differ from many of the studies prior those published before 2000 that gave rise to the laws of software evolution. But the stage is set for how to proceed in pursuing the ontological foundation of the laws and theory of software evolution.

On the other hand, if the theory of feedback control systems becomes too complicated or too rigid of an ontological framework for describing and explaining software evolution, then alternative ontological frameworks may be employed to further such study.

5.2 Alternative Ontologies for F/OSS Evolution

One observation from studying the evolution of technical systems is that the technologies and techniques for developing and maintaining F/OSS constitute a distinct technological regime. This regime for F/OSS is not anticipated or adequately covered by other studies of software evolution. The same may also be true of emerging technologies like component-based software systems and those with dynamically composed run-time architectures. Thus, it seems that any ontology for software evolution should account for the emergence, deployment, and consequences of use for new tools, techniques, and concepts for software development, as well as the productive units, technical system infrastructure, and technological regime in they are situated.

A second observation from the study of the evolution of F/OSS is that different types of software system evolve at substantially different rates--some super-linear, some constant, some sub-linear, some not at all. Small software systems may not evolve or thrive for very long, nor will they be assimilated into larger systems, unless merged with other systems whose developers can form a critical mass sufficient to co-evolve with the composite system and productive unit. Drawing from biological evolutionary theory, it may be that software evolution theory requires or will benefit from *taxonomic analyses* to describe, classify and name different types of software systems or architectural morphologies, thus refactoring the conceptual space for software system evolution. Similarly, it may benefit from *phylogenetic analyses* that reconstruct the evolutionary histories of different types of software systems, whether as open source and closed source implementations. Last, it suggests that a science of *software systematics* is needed to encourage study of the kinds and diversity of software programs, components, systems, and application domains, as well as relationships among them, across populations of development projects within different technological regimes over time. This would enable comparative study of contemporary software systems with their ancestral lineage,

as well as to those found within the software fossil record (e.g., those software systems developed starting in the 1940's onward for mainframe computers, and those developed starting in the 1970's for personal computers). Finally, this could all be done in ways that enable free/open source computational modeling of such a framework for software evolution.

A third observation from the emergence and evolution of F/OSS is that the beliefs, narratives, and memes play a role in facilitating the adoption, deployment, use and evolution of F/OSS. Their role may be more significant than the cultural and language constructs that accompanied the earlier technological regime of centralized, closed source software development that primarily developed systems for deployment in corporate settings. Similarly, relatively new software language constructs for scripting, plug-in modules, and extensible software architectures have been popularized in the regime of F/OSS. But these constructs may also have enabled new forms of architectural evolution and bricolage, thereby accelerating the growth rate of large F/OSS in a manner incommensurate to that seen in the world of mainframe software systems, an earlier technological regime. Finally, large and popular F/OSS systems are being extended and evolved to accommodate end-users and developers whose native language or ethnic legacy is not English based. The internationalization or localization of F/OSS systems, while neither necessarily adding nor subtracting functionality, does create value in the global community by making these systems more accessible to a larger audience of prospective end-users, developers, reviewers and debuggers. These software extensions add to the bulk of F/OSS code release size in probably orthogonal ways, but may or may not represent anti-regressive work [cf. Lehman, Ramil, and Kahen 2001].

A fourth observation from the evolution of F/OSS is that they have emerged within a technological regime where competitive market forces and organizational ecologies surrounding closed source software systems may have effectively served to stimulate the growth and diffusion of F/OSS project populations. Furthermore, it may be the case that these circumstances are co-evolving with the relative growth/demise of open versus closed source software product offerings, and the communities of developers who support them. Other studies of software evolution make little/no statement about the effects of market forces, competition, organizational ecology, co-evolution, or the spread of software project populations as contributing factors affecting how software systems may evolve. Yet many of the largest F/OSS systems are pitted directly against commercially available, closed source alternatives. These F/OSS systems typically compete against those developed within centrally controlled and resource managed software development centers. Thus, it seems appropriate to address how co-evolutionary market forces surround and situate the centralized or decentralized organizational ecologies that develop and maintain large software systems in order to better understand how they evolve.

A last observation from a view of F/OSS as a socio-technical world is that the evolution of F/OSS system is situated within distinct web of organizational, technological, historical and geographic contexts. However, feedback control systems typically do not account for organizational productive units or their historical circumstances. Similarly, there is no accounting for the motivations, beliefs, or cultural values of software developers who may prefer software systems to be developed in a manner that is free and

open, so as to enable subsequent study, learning, reinvention, modification, and redistribution. But as seen above, these are plausible variables that can contribute to the evolution of F/OSS, and thus further study is required to understand when, where and how they might influence how particular F/OSS systems may evolve.

6. Conclusions

The laws and theory of software evolution proposed by Lehman and colleagues are recognized as a major contribution to the field of software engineering and the discipline of computer science. These laws have been generally found to provide a plausible explanation for how software systems evolve throughout their life. They have been explored empirically over a period of more than 30 years, so their persistence is a noteworthy accomplishment. Developing laws and theory of software evolution relying on empirically grounded studies is a long-term endeavor that poses many challenges in research method, theoretical sampling of systems to study, theory construction, and ongoing theory testing, refutation, and refinement. However, it may prove to be an endeavor that gives rise to new ways and means for conceptualizing evolutionary processes in other domains of study.

As the technology, process, and practice of software development and maintenance has evolved, particularly in the past ten years and with the advent of large numbers of free/open source software development projects, it has begun clear that the existing models of software evolution based on empirical studies of closed source systems prior to 2000 may be breaking down, at least from results of the many empirical studies of F/OSS reviewed in this chapter. The models and prior studies do not address and therefore do not provide a rich or deep characterization of the evolution of F/OSS systems. Prior models of software evolution were formulated in the context of software development and maintenance processes and work practices that were based in centralized, corporate software development centers that built large closed source system applications with few competitive offerings for use by large enterprises. Large F/OSS systems, on the other hand, are developed and maintained in globally decentralized settings that collectively denote a loosely-coupled community of developers/users who generally lack the administrative authority, resource constraints, and schedules found in centrally controlled software centers. These F/OSS systems are typically competing alternatives to closed source commercial software product offerings. Subsequently, it may be better to consider whether the primary evolutionary dynamic associated with F/OSS is reinvention, renovation, or revitalization of established software systems or applications that have proved to be useful, but now merit redevelopment, refinement, and new extensions or extension mechanisms [Scacchi 2004]. Similarly, as large F/OSS are sometimes observed to exhibit sustained super-linear or exponential growth, can such rates of growth go on unabated, or will the concurrent growth of system complexity eventual change the shape of the growth curve to something more like an “S” curve, with exponential growth in the early stages, followed by inverse-square growth in the later stages [cf. Lehman and Ramil 2002]. Further study of such matters is clearly needed.

There is a growing base of data, evidence, and findings from multiple studies of F/OSS systems that indicate F/OSS systems co-evolve with their user-developer communities, so that growth and evolution of each depends on the other. Co-evolution results of this kind are not yet reported for closed source systems, and it is unclear that such results will be

found. In short, prior models of software evolution were developed within and apply to systems maintained and used in a corporate world and technological regime that differs from the socio-technical communities, global information infrastructure, and technological regime which embeds open source software.

It appears that we need a more articulate explication and refinement of models of software evolution if they are to account for the evolution of F/OSS systems. One way this might be done is to embrace and extend reliance of the ontology of feedback control systems theory. This would entail identifying the types, operations, behaviors, and interconnection of mechanisms that embody and realize a complex, multi-level, multi-loop, and multi-agent feedback system. Building computational models and simulations of such a system (or family of systems) could be a significant contribution. Otherwise, alternative evolutionary ontologies might be adopted, individually or in some explicit hybrid combination form. The choice of which ontology to use will suggest the types of entities, flows, mechanisms, and controls for software evolution should be modeled, measured, improved, and refined according to some conceptual or theoretically motivated framework. Otherwise, use of alternative ontologies may accommodate new models of theories of software evolution that do not rely on high-level, abstract or over-generalized models, but instead may result in theories or models of smaller and more precise scope that better account for the complex, socio-technical ecological niches where software systems evolve in practice, as well as for the type and history of the system in such context.

Theories of software evolution should be empirically grounded. They should be formulated or modeled in ways in which they can be subject to tests of refutation or refinement. The tests in turn should examine comparative data sets that are theoretically motivated, rather than motivated by the convenience of data at hand that may have been collected and conceived for other more modest purposes. There should be theories that address software evolution within, as well as, across generations of software technology or technological regimes. Laws and theories of software evolution should have a computational rendering so that their source code, internal representation, and external behavior can be observed, shared, studied, modified and redistributed. They should be free (as in *libre*) and open source. These models should then also be suitable for simulation, analysis, visualization, prototyping, and enactment [Scacchi 2002b, Scacchi and Mi 1997]. By doing this, the software engineering and computer science community can make a new contribution in the form of reusable assets that can be adopted and tailored for use in other domains of evolution theorizing.

The future of research in software evolution is must include the technological regime of F/OSS as a major element. This will be an increasingly practical choice for empirical study of individual systems, groups of systems of common type, and of larger regional or global populations of systems. This is due in part to the public availability of the source code and related assets on the Web for individual versions/releases of hundreds of application systems, as well as data about their development processes, community participants, tools in use, and settings of development work. Not that collecting or accessing this data is without its demands for time, skill, effort and therefore cost, but that useful and interesting data can be accessed and shared without the barriers to entry and corporate disclosure constraints of intellectual property claims or trade secrets. It seems

unlikely that the software engineering community will get open access to the source code, bug report databases, release histories, or other "property or secrets" of closed source systems that are in widespread use (e.g., Microsoft Windows operating systems, Internet Explorer, Word, Outlook, Office, Oracle DBMS, or SAP R/3) in ways that can be shared and studied without corporate trade secrets, non-disclosure agreements and publication constraints. In contrast, it is possible today to empirically study the ongoing evolution of the GNU/Linux operating systems (Kernel or alternative distributions), the Mozilla Web browser, Open Office, SAP DB, Apache project, or GNUenterprise, which together with their respective technically and socially networked communities, have publicly accessible Web portals and software assets that can be shared, studied, and redistributed to support research into models, laws, and theory of software evolution. The future of research in software evolution should be free, open and constructive, since it will likely take a community of investigators to help make substantial progress in developing, refining, sharing, and publishing models, laws, and theories of software evolution.

7. Acknowledgements

The research described in this report is supported by grants from the National Science Foundation #ITR-0083075, #ITR-0205679, #ITR-0205724 and #ITR-0350754. No endorsement implied. Mark Ackerman at the University of Michigan Ann Arbor; Les Gasser at the University of Illinois, Urbana-Champaign; John Noll at Santa Clara University; Margaret Elliott, Mark Bergman, Chris Jensen and Xiaobin Li at the UCI Institute for Software Research; and Julia Watson at The Ohio State University are also collaborators on the research project from which this article was derived. Finally, Manny Lehman, Nazim Madhavji, and Juan Ramil provided many helpful comments, suggestions, and clarifications on earlier versions of this chapter.

8. References

W.J. Abernathy, *The Productivity Dilemma: Roadblock to Innovation in the Automobile Industry*, John Hopkins University Press, 1978.

T. Abdel-Hamid and S. E. Madnick, *Software Project Dynamics: An Integrated Approach*. Prentice Hall Software Series, New Jersey, 1991.

R.N. Bateson, *Introduction to Control System Technology*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

A. Bauer and M. Pizka, The Contribution of Free Software to Software Evolution, *Proc. Sixth International Workshop on Principles of Software Evolution (IWPSE'03),* 170-179, Helsinki, Finland, September 2003.

S. Beecham, T. Hall, and A. Rainer, Software Process Improvement Problems in Twelve Software Companies: An Empirical Analysis, *Empirical Software Engineering*, 8(1), 7-42, 2003.

S. Bendifallah and W. Scacchi, Understanding Software Maintenance Work, *IEEE Trans. Software Engineering*, 13(3), 311-323, March 1987. Reprinted in D. Longstreet (ed.), *Tutorial on Software Maintenance and Computers*, IEEE Computer Society, 1990.

B.E. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.

P.J. Bowler, *Evolution: The History of an Idea* (Revised Edition), University of California Press, Berkeley, CA, 1989.

A. Capilupppi, P. Lago, and M. Morisio, Characteristics of Open Source Projects, *Proc. 7th European Conference on Software Maintenance and Reengineering*, March 2003

M. Christiansen and S. Kirby (eds.), *Language Evolution: The States of the Art*, Oxford University Press, 2003.

R. Conradi and A. Fuggetta, Improving Software Process Improvement, *IEEE Software*, 92-99, July-August 2002.

S. Cook, H. Ji and R. Harrison, Software Evolution and Software Evolvability, unpublished manuscript, University of Reading, UK, 2000.

K. Crowston, H. Annabi, and J. Howison, Defining Open Source Software Project Success, *Proc. Intern. Conf. Information Systems (ICIS 2003)*, Seattle, WA, 327-340, December, 2003.

M.A. Cusumano and D.B. Yoffie, Software Development on Internet Time, *Computer*, 60-70, October 1999.

C. DiBona, S. Ockman and M. Stone, *Open Sources: Voices from the Open Source Revolution*, O'Reilly Press, Sebastopol, CA 1999.

M. Di Penta, M. Neteler, G. Antonio, and E. Merlo, Knowledge-Based Library Refactoring for an Open Source Project, *Proc. IEEE Working Conf. Reverse Engineering*, Richmond VA, October 2002.

J.C. Doyle, B.A Francis and A.R. Tannenbaum, *Feedback Control Theory*, Macmillan, New York, 1992.

J. Erenkrantz, Release Management within Open Source Projects, *Proc. 3rd. Workshop on Open Source Software Engineering*, 25th Intern. Conf. Software Engineering, Portland, OR, May 2003.

S.G. Eick, T.L. Graves, A.F. Karr, J.S. Marron, and A. Mockus, Does Code Decay? Assessing the Evidence from Change Management Data, *IEEE Trans. Software Engineering*, 27(1), 1-12, January 2001.

M. Elliott and W. Scacchi, *Free Software Development: Cooperation and Conflict in a Virtual Organizational Culture*, in S. Koch (ed.), *Free/Open Source Software Development*, Idea Press, 2004 (to appear).

L. Gabora, The Origin and Evolution of Culture and Creativity, *Journal of Memetics - Evolutionary Models of Information Transmission*, 1, 1997. http://jom-emit.cfpm.org/vol1/gabora_l.html

L. Gabora, The Beer Can Theory of Creativity, in P. Bentley and D. Corne (eds.) *Creative Evolutionary Systems*, Morgan Kaufman, 2000.

H. Gall, M. Jayazeri, R. Kloesch and G. Trausmuth, Software Evolution Observations Based on Product Release History, *Proc. 1997 Intern. Conf. Software Maintenance (ICSM'97)*, Bari, IT, October 1997.

B. Glaser and A. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*, Aldine Publishing, Chicago, IL, 1976.

M.W. Godfrey and E.H.S. Lee, Secrets from the Monster: Extracting Mozilla's Software Architecture, *Proc. Second Intern. Symp. Constructing Software Engineering Tools* (CoSET-00), Limerick, Ireland, June 2000.

M.W. Godfrey and Q. Tu, Evolution in Open Source Software: A Case Study, *Proc. 2000 Intern. Conf. Software Maintenance* (ICSM-00), San Jose, California, October 2000.

J.M Gonzalez-Barahona, L. Lopez, and G. Robles, Community Structure of modules in the Apache project, *Proc. 4th Workshop on Open Source Software Engineering*, Edinburgh, Scotland, May 2004.

J.M Gonzalez-Barahona, M.A. Ortuno Perez, P. de las Heras Quiros, J. Centeno Gonzalez, and V. Matellan Olivera, Counting Patotoes: The Size of Debian 2.2, *Upgrade Magazine*, II(6), 60-66, December 2001.

S.J. Gould, *The Structure of Evolutionary Theory*, Harvard University Press, Cambridge, MA, 2002.

M.T. Hannan and G.R. Carroll, *Dynamics of Organizational Populations: Density, Legitimation and Competition*, Oxford University Press, New York, 1992.

A. Hars and S. Ou, Working for Free? Motivations for Participating in Open-Source Software Projects, *Intern. J. Electronic Commerce*, 6(3), 25-39, 2002.

T.J. Hughes, The Evolution of Large Technological Systems, in W. Bijker, T. Hughes, and T. Pinch (eds.), *The Social Construction of Technological Systems*, MIT Press, Cambridge, MA, 51-82, 1987.

F. Hunt and P. Johnson, On the Pareto Distribution of SourceForge Projects, in C. Gacek and B. Arief (eds.), *Proc. Open Source Software Development Workshop*, 122-129, Newcastle, UK, February 2002.

C. Jensen and W. Scacchi, Simulating an Automated Approach to Discovery and Modeling of Open Source Software Development Processes, *Proc. 4th Software Process Simulation and Modeling Workshop (ProSim'03),* Portland, OR, May 2003.

C. Jensen and W. Scacchi, Process Modeling Across the Web Information Infrastructure, *Proc. 5th Software Process Simulation and Modeling Workshop (ProSim'04),* Edinburgh, Scotland, UK, May 2004.

J.L. King and K.L. Kraemer, Evolution and Organizational Information Systems: An Assessment of Nolan's Stage Model, *Communications ACM*, 27(5), 466-475, 1984.

C.F. Kemerer and S. Slaughter, An Empirical Approach to Studying Software Evolution, *IEEE Trans. Software Engineering*, 25(4), 493-505, 1999.

G. Kniesel, J. Noppen, T. Mens, and J. Buckley, *WS 9. The First International Workshop on Unanticipated Software Evolution,* Workshop Report, Malaga, Spain, June 2002. http://joint.org/use2002/ecoopWsReportUSE2002.pdf

S. Koch and G. Schneider, Results from Software Engineering Research into Open Source Development Projects Using Public Data, *Diskussionspapiere zum Tätigkeitsfeld Informationsverarbeitung und Informationswirtschaft,* Hans R. Hansen und Wolfgang H. Janko (Hrsg.), Nr. 22, Wirtschaftsuniversität Wien, 2000.

I. Lakatos, *Proofs and Refutations: The Logic of Mathematical Discovery*, Cambridge University Press, Cambridge, UK, 1976.

M.M. Lehman, Programs, Life Cycles, and Laws of Software Evolution, *Proc. IEEE*, 68, 1060-1078, 1980.

M.M. Lehman, Rules and Tools for Software Evolution Planning and Management, in J. Ramil (ed.), *Proc. FEAST 2000*, Imperial College of Science and Technology, London, 53-68, 2000. Also appears with J.F. Ramil in an expanded version as "Rules and Tools for Software Evolution Management," in *Annals of Software Engineering*, 11, 16-44, 2001.

M.M. Lehman, Software Evolution, in J. Marciniak (ed.), *Encyclopedia of Software Engineering*, 2nd Edition, John Wiley and Sons Inc., New York, 1507-1513, 2002. Also see "Software Evolution and Software Evolution Processes," *Annals of Software Engineering*, 12, 275-309, 2002.

M.M. Lehman and L.A. Belady, *Program Evolution – Processes of Software Change*, Academic Press, London, 1985.

M.M. Lehman, D.E. Perry and J.F. Ramil, Implications for Evolution Metrics on Software Maintenance, *Proc. 1998 Intern. Conf. Software Maintenance (ICSM'98),* Bethesda, MD, 1998.

M.M. Lehman and J.F. Ramil, An Approach to a Theory of Software Evolution, *Proc. 2001 Intern. Workshop on Principles of Software Evolution*, 2001.

M.M. Lehman and J.F. Ramil, An Overview of Some Lessons Learnt in FEAST, *Proc. Eighth Workshop on Empirical Studies of Software Maintenance* (WESS'02), Montreal, CA, 2002.

M.M. Lehman and J.F. Ramil, Software Evolution, in this volume, 2004.

M.M. Lehman, J.F. Ramil, and G. Kahen, *A Paradigm for the Behavioural Modelling of Software Processes using System Dynamics*, technical report, Dept. of Comp., Imperial College, London, September 2001.

Lehman MM, Ramil JF and Sandler U, *An Approach to Modelling Long-Term Growth Trends in Software Systems*, Proc. ICSM 2001, 6 - 10 Nov., Florence, Italy

M.M. Lehman, J.F. Ramil, P.D. Wernick, D.E. Perry, and W. Turski, Metrics and Laws of Software Evolution – The Nineties View, *Proc. 4th Intern. Symp. Software Metrics*, 20-32, Albuquerque, NM, November 1997.

G. Madey, V. Freeh, and R. Tynan, The Open Source Software Development Phenomenon: An Analysis Based on Social Network Theory. *Proc. Americas Conference on Information Systems (AMCIS2002)*. 1806-1813, Dallas, TX, 2002.

T. Mens, J. Buckley, M. Zenger, and A. Rashid, Towards a Taxonomy of Software Evolution, *Second Intern. Workshop on Unanticipated Software Evolution*, Warsaw, Poland, April 2003. http://joint.org/use2003/Papers/18500066.pdf

M.H. Meyer and J.M. Utterback, The Product Family and the Dynamics of Core Capability, *Sloan Management Review*, 34(3), 29-47, Spring 1993.

P. Mi and W. Scacchi, A Knowledge-Based Environment for Modeling and Simulating Software Engineering Processes, *IEEE Trans. Data and Knowledge Engineering*, 2(3), 283-294, September 1990. Reprinted in *Nikkei Artificial Intelligence*, 20(1), 176-191, January 1991 (in Japanese); also in *Process-Centered Software Engineering Environments*, P.K. Garg and M. Jazayeri (eds.), IEEE Computer Society, 119-130, 1996.

P. Mi and W. Scacchi, Process Integration in CASE Environments, *IEEE Software*, 9(2), 45-53, March 1992. Reprinted in Eliot Chikofsky (ed.), *Computer-Aided Software Engineering (CASE)*, Second Edition, IEEE Computer Society, 1993.

P. Mi and W. Scacchi, A Meta-Model for Formulating Knowledge-Based Models of Software Development, *Decision Support Systems*, 17(4), 313-330, 1996.

A. Mockus, R.T. Fielding, and J. Herbsleb, Two Case Studies of Open Source Software Development: Apache and Mozilla, *ACM Trans. Software Engineering and Methodology*, 11(3), 309-346, 2002.

K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y.Ye, Evolution Patterns of Open-Source Software Systems and Communities, *Proc. 2002 Intern. Workshop Principles of Software Evolution*, 76-85, 2002.

B. Nardi and V. O'Day, *Information Ecologies: Using Technology with Heart*, MIT Press, Cambridge, MA 1999.

R.R. Nelson and S.G. Winter, *An Evolutionary Theory of Economic Change*, Belknap Press, Cambridge, MA, 1982.

M.H. Nitecki, (ed.), *Evolutionary Progress*, University of Chicago Press, Chicago, IL, 1988.

S. O'Mahony, Developing Community Software in a Commodity World, in M. Fisher and G. Downey (eds.), *Frontiers of capital: Ethnographic Reflections on the New Economy*, Social Science Research Council, (to appear), 2003.

J.W. Paulson, G. Succi, and A. Eberlein, An Empirical Study of Open-Source and Closed-Source Software Products, *IEEE Trans. Software Engineering*, 30(4), 246-256, April 2004.

D.E. Perry, and J.F. Ramil, Empirical Studies of Software Evolution, in this volume, 2004.

D.E. Perry, H.P. Siy, and L.G. Votta, Parallel Changes in Large-Scale Software Development: An Observational Case Study, *ACM Trans. Software Engineering and Methodology*, 10(3), 308-337, 2001.

K.R. Popper, *Conjectures and Refutations*, Routledge & Kagen, 1963.

J. F.Ramil, *Laws of Software Evolution and their Empirical Support*, Invited Panel Statement, Proc. ICSM 2002, Montreal, 3-6 Oct 2002, p. 71

C.R. Reis and R.P.M. Fortes, An Overview of the Software Engineering Process and Tools in the Mozilla Project, *Proc. Workshop on Open Source Software Development*, 155-175, Newcastle, UK, February 2002.

G. Robles-Martinez, J.M. Gonzalez-Barahona, J. Centeno Gonzalez, V. Matellan Olivera, and L. Rodero Merino, Studying the Evolution of Libre Software Projects using Publicly Available Data, *Proc. 3rd Workshop on Open Source Software Engineering*, Portland, OR, 2003.

P.P. Saviotti and G.S. Mani, Competition, Variety and Technological Evolution: A Replicator Dynamics Model, *J. Evolutionary Economics*, 5(4), 369-92, 1995.

W. Scacchi, Understanding Software Productivity: Towards a Knowledge-Based Approach, *Intern. J. Software Engineering and Knowledge Engineering*, 1(3), 293-321,

1991. Revised version in D. Hurley (ed.), *Advances in Software Engineering and Knowledge Engineering*, Volume 4, 37-70, 1995.

W. Scacchi, Experiences in Software Process Simulation and Modeling, *J. Systems and Software*, 46(2/3), 183-192, 1999

W. Scacchi, Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings – Software*, 149(1), 24-39, February 2002a.

W. Scacchi, Process Models for Software Engineering, in J. Marciniak (ed.), *Encyclopedia of Software Engineering*, 2nd Edition, John Wiley and Sons Inc., New York 993-1005, 2002b.

W. Scacchi, *Open EC/B: A Case Study in Electronic Commerce and Open Source Software Development*, technical report, Institute for Software Research, July 2002c.

W. Scacchi, Free/Open Source Software Development in the Game Community, *IEEE Software*, 21(1), 59-67, January/February 2004.

W. Scacchi and P. Mi, Process Life Cycle Engineering: Approach and Support Environment, *Intern. J. Intelligent Systems in. Accounting, Finance, and Management*, 6:83-107, 1997.

S.R. Schach, B. Jin, D.R. Wright, G.Z. Heller, and A.J. Offutt, Maintainability of the Linux Kernel, *IEE Proceedings – Software*, 149(1), 18-23, February 2002.

N. Smith and J.F. Ramil, Qualitative Simulation of Software Evolution Processes, *WESS'02 Eighth Workshop on Empirical Studies of Software Maintenance*, Montreal, October 2002.

N. Smith, A. Capiluppi, and J.F. Ramil, Qualitative Analysis and Simulation of Open Source Software Evolution, *Proc. 5th Software Process Simulation and Modeling Workshop (ProSim'04)*, Edinburgh, Scotland, UK, May 2004.

M. Svahnberg and J. Bosch, Evolution in Software Product Lines, *J. Software Maintenance*, 11(6), 391-422, 1999.

T. Tamai and Y. Torimitsu, Software Lifetime and its Evolution Process over Generations, *Proc. Conf. Software Maintenance*, 63-69, November 1992.

W. Turski, Reference Model for Smooth Growth of Software Systems, *IEEE Trans. Software Engineering*, 22(8), 599-600, August 1996.

J.M. Utterback, *Mastering the Dynamics of Innovation: How Companies Can Seize Opportunities in the Face of Technological Change*, Harvard Business School Press, 1994.

N.M. Victor and J.H. Ausubel, DRAMs as Model Organisms for Study of Technological Evolution, *Technological Forecasting and Social Change*, 69(3), 243-262, April 2002.

J. van den Ende and R. Kemp, Technological transformations in history: how the computer regime grew out of existing computing regimes, *Research Policy*, 28, 833-851, 1999.

E. von Hippel and R. Katz, Shifting Innovation to Users via Toolkits, *Management Science*, 48(7), 821-833, July 2002.

R. Yin, *Case Study Research: Design and Methods (Second Edition)*, Sage Publications, Newbury Park, CA. 1994.