Visualizing Historical Data Using Spectrographs

Ahmed E. Hassan, Jingwei Wu, and Richard C. Holt
Software Architecture Group (SWAG)
School of Computer Science
University of Waterloo
Waterloo, Canada

{aeehassa,j25wu,holt}@plg.uwaterloo.ca

ABSTRACT
Studying the evolution of long lived processes such as the development history of a software system or the publication history of a research community, requires the analysis of a vast amount of data. Aggregation techniques and data specific techniques are usually used to cope with the large amount of data.

In this paper, we introduce a general technique to study historical data derived from tracking the evolution of long lived processes. We present a visualization approach (*evolution spectrographs*) to assist in identifying interesting patterns and events during evolutionary analysis of such historical data. We demonstrate the usefulness of spectrographs through several case studies. The data for the case studies are derived from the publication history of conferences in the area of software engineering and from the source control of several large open source projects. Our case studies reveal interesting patterns such as the increase of collaboration over time in the area of software engineering, and the emergence of new research topics. The spectrographs give an overview of the change activities for the subsystems in large software projects.

1 INTRODUCTION

To study the evolution of a software system or a research community, one is likely to measure a particular *property* of the studied *data entities* at specific *time intervals*. Table 1 shows examples of possible evolutionary studies. For example, we can track the number of new authors in a research community each year (example 1), track the number of files changed in each release of a software system (example 2), or measure the average number of functions that are changed for each changed function on a release basis (example 3).

Example#	Property	Data Entities	Time Interval
1	New	Authors	Yearly
2	Changed	Files	Release
3	Average co-changed	Functions	Release

Table 1: Examples of Evolutionary Studies

For large data such as the data generated from monitoring the evolution of long lived projects or communities, researchers tend to use techniques which could reduce the amount of data at hand in order to ease the required analysis. Two aggregation techniques are usually used:

Coarser Time Granularity (*Fusing*): Researchers may study the major releases of a software system instead of studying all its minor releases or nightly builds. By fusing data for consecutive releases or time periods, researchers need to analyze a smaller number of data points which correspond to fewer time intervals or releases.

Coarser System Granularity (*Lifting*): Researchers are likely to measure properties of high level entities. For example, instead of studying the evolution of the size of each function or file in a software system, researchers are more likely to measure changes to the size of the whole software system. By lifting data from low level entities such as function to high level entities such as subsystem or the overall system, researchers need to analyze the evolution of a smaller number of entities. The smaller number of entities permits the use of less sophisticated techniques such as simple metric plots ($e.g.$ [11]).

Unfortunately, both aggregation techniques have their shortcomings:

Coarser Time Granularity (*Fusing*): The reduction of the time granularity is likely to cause the appearance of artificial spikes in the evolution data. For example, the size of a software system is likely to show large increases from one release to the next if the data is studied at the release level, due to the long time periods between releases. Such sudden increases may not show up if the data is studied on a monthly basis. Furthermore, the reduction of the time granularity may cause the disappearance of interesting events in the data, for example the removal of a small subsystem (drop in size) and its replacement with a much larger subsystem (increase in size) would simply show up as a slight increase in the size of the overall system and the disappearance of the smaller subsystem would gone unnoticed.

Coarser System Granularity (Lifting): The lifting of the data may hide interesting evolutionary details due to

the canceling out of low level patterns or details. Such shortcoming was demonstrated by Gall $et\ al.$ and Godfrey $et\ al.$ Gall $et\ al.$ studied the evolution of an industrial telephony system and discovered that a number of subsystems exhibited interesting evolutionary patterns which were canceled out when the system was examined as a whole [7]. Similarly, Godfrey $et\ al.$ examined the evolution of the Linux operating system. He uncovered at first that the system as a whole was growing at a fast (likely unsustainable) rate; a deeper look (examining each subsystem) indicated that the fast growth is occurring mainly in the hardware drivers subsystems which are self contained entities which do not interact [13].

Although aggregation techniques have their shortcomings, they are still needed to cope with the complexities associated with studying and analyzing the evolution of large data sets [14].

In this paper we are interested in striking a balance between too much aggregation (fusing and lifting) and more detailed analysis. We present a visualization approach (*evolution spectrographs*) to study large historical data sets at a reasonable granularity. The approach employs a special coloring technique to ease the identification of interesting evolutionary patterns.

To showcase the applicability and the generality of our approach, we use spectrographs to examine the evolution of several types of data, such as the publication records in software engineering conferences and the change data stored in source control systems for several open source projects. The spectrographs reveal interesting information about the large data such as:

1. The emergence of new research topics in the area of software engineering.
2. The collaboration patterns between researchers and developers.
3. The shift of focus of software development teams working on large projects over time.
4. The emergence of hidden dependencies between subsystems over time.

Organization Of Paper

The paper is organized as follows. Section 2 describes the data used in our evolutionary analysis case studies. Section 3 overviews our visualization technique (*spectrographs*) which is used to analyze the aforementioned data. We discuss the coloring technique employed to ease the identification of patterns in large historical data. Section 4 presents our case studies and reports on the main findings of each study. Section 5 describes prior work which dealt with studying and visualizing evolutionary patterns in large data sets. Section 6 concludes the paper with parting thoughts about the main contributions of the presented work, results of the case studies, and an overview of some of the limitations and strength of our approach.

2 Studying Evolution

Studying the evolutionary process usually consists of measuring *properties* of the studied *data entities* at specific *time intervals*. In our evolutionary study, we used a variety of data sets. To permit us to reuse our analysis techniques for the different types of data, we map each data set into a canonical format and we perform our analysis on the canonical format that is data independent. In this section, we present the studied data, along with the entities and properties we chose to study. We also discuss our mapping to permit the reuse of analysis across data sets.

Studied Data

We used two sources of historical data in our analysis:

1. Data derived from the publication records of researchers in the area of software engineering.
2. Data recovered from the repositories of source control systems for several large open source projects.

Both data sources are rather large and are likely to contain several interesting patterns since they are based on the evolution of complex and long lived processes (software development and research publication processes). We now present details about both data sources and our motivation for studying the data.

Publication Records

Publications in a research community give a picture of the progress of collaboration and emergence of topics in an active research field. The authorship details on each publication represent a social network of collaboration between researchers (authors) in the community. One would hope to have a high degree of collaboration in an academic community, in comparison to commercial communities.

Furthermore, the title of each publication can give us a picture of the emergence of research topics and areas of interest in the community over time. A good understanding of the progress of research interest in a community is likely to shed some light on its evolution.

The DBLP [3] tracks the publication history for several computer science conferences. The data is available as an XML file. It records the title of the publications and the authors of these publications. Using this data, we can investigate several evolutionary phenomena. For example, we could visualize the collaboration of researchers in the field to understand how collaboration between researchers has evolved as the software engineering field matured. We can also monitor the emergence and disappearance of research topics that have shaped research in the field throughout time.

We analyzed the publication records of 21 software engineering conferences from the DBLP database. The 21 conferences chosen by us are: ADSD, APSEC, ASE, KBSE,

Conferences	21
Authors	9,994
Papers	7,407
Years	25

Table 2: Statistics about the DBLP Data for the Software Engineering Field

CAiSE, COMPSAC, COOTS, ECOOP, ESEC, FSE, ICSE, ICSR, METRICS, OOPSLA, PASTE, RE, SEKE, ICSM, WCRE, IWPC, and CSMR. Table 2 shows descriptive statistics about the publication data. Due to the size of the data, aggregation techniques such averaging are likely to be used. We would like to reduce the need for aggregating the data. We propose the evolution spectrograph approach, covered in the following section, to gain a more detailed understanding of the data while reducing the need for aggregating the data.

Source Control Data
Whereas publication records track the evolution of a research field and collaboration between its researchers. Data stored in source control repositories track the progress of software projects and the interaction between its developers and the source code.

Source control systems are used extensively by large software projects to control and manage their source code [18]. Data stored in these repository presents a great opportunity to study the code development and change process. The data collection costs are minimal since it is collected automatically as modifications are done to the source code.

The repository of a source control system contains various details about the development history of every file in a project. It contains the creation date of a file, its initial content and a record of every modification done to the file. A *modification record* stores the date of the modification, the name of the developer who performed the changes, the number of lines that were changed, the actual lines of code that were added or removed, and a detailed message entered by the developer explaining the reasons for the change. Using the time of the change, the detailed message, and the developer's name, we can determine all files that changed together ($i.e.$ *Changelist*) to implement or enhance a particular feature, or fix a specific bug. For the results presented in this paper, we do not consider in our analysis changelists which correspond to book keeping changes. Book keeping changelists are changelists where more than 15 files are changed together or which have words such as 'merge', 'clean up', or 'update copyright' in their detailed message.

Unifying the Data
A straightforward approach to study the publication data is to create a social collaboration network for the authors of papers in the data [9]. We create a node for each author that published a paper, and we create an edge between two authors if they co-authored a paper together. We assign a weight to each node (author) that is based on the number of papers written by the author.

At first glance, it may seem that the DBLP publication records and the source control modification records do not have much in common. It may seem that each data would require specialized analysis techniques to study its evolution. Luckily, this is not the case. We map entities and relations in both data sets into a canonical format. Using this format, we could use the same techniques to study both data sets.

The same aforementioned approach of creating a social collaboration network could be used to create a network for the words in the paper titles in the publication data. Using the paper titles, we removed stopwords (such as "the" and "of") and used a stemmer to derive the root of each word in the title of a paper (for example, truncating "extracting" to "extract"). A node is created for each stemmed word and an edge is created between two nodes if they both existed in the same paper title. The weight of a node is proportional to the number of papers that have the word corresponding to that node in their title.

Similarly, we could create two networks form the source control modification records. We could create a developer collaboration network. Each developer is assigned a node and an edge exists between two nodes if they both modified a common subsystem in the software system. We could also create a subsystem collaboration network. Each subsystem is assigned a node and an edge exists between two nodes if files that existed in both subsystems were modified as part of the same *Changelist*. Table 3 summarizes the different networks that we created for our analysis. We could create other networks to study other evolutionary aspects of the data, nevertheless we focus only on these four networks in this paper.

Studied Properties
Once the historical data is mapped to the canonical network, we can study properties of the network that are independent of the data. We can later express our results in the context of the historical data. We study two properties of a network:

1. The growth of the weight of each node.
2. The growth of the number of neighbors of each node.

Both these properties are likely to reveal interesting information and patterns about changes to the data. For example, in a research community it is preferable that the collaboration between authors increases over time. In other words, we would expect that the number of neighbors for a node to increase over time instead of remaining constant. We now explore both properties and the metrics we chose to measure them.

Weight Growth
We measure the growth of the weight of each node in com-

Network	Node	Edge	Node Weighting	Source
Author	Author	Co-authored a paper together	Number of papers written by author	DBLP data
Word	Stemmed word	Words occur in a paper title together	Number of papers with word in title	DBLP data
Developer	Developer	Modified common subsystems	Number of changes done by developer	CVS data
Subsystem	Subsystem	Changed in the same Change-list	Number of changes done to subsystem	CVS data

Table 3: Description of the Created Networks from the Publication Records and Source Control Data Sets

Network	**Changes to W for a node x indicate that**
Author	Author x has contributed a large number of publications in the studied research field during the time period.
Word	Word x is becoming a popular buzzword or research is focusing on the area related to word x during the time period.
Developer	Developer x has implemented a large percentage of the changes and x is likely to be the most knowledgable person about the changes that occurred to the software system during the time period
Subsystem	Subsystem x has been the focus of most changes during the time period.

Table 4: Possible Explanations for The Changes to the W Metric Based on The Network Type.

Network	**Changes to N for a node x indicate that**
Author	Author x has collaborated with a larger than average collaborators.
Word	Word x is being used more often in newer contexts than it used to be in the past.
Developer	Developer x is likely to be interacting more with other developers on the team or gaining a better knowledge of other developers' coding styles as she/he works on subsystems that have been touched by other developers.
Subsystem	Subsystem x may be becoming too dependant on other subsystems due to the new co-change dependencies.

Table 5: Possible Explanations for The Changes to the N Metric Based on The Network Type.

parison to the total growth of the weight of all nodes in the network during each time period. A formal definition of the metric used to study the weight growth property (W) for a node x during period t:

$$W(x) = \frac{change\ to\ weight\ of\ node\ x\ during\ period\ t}{total\ change\ to\ weight\ of\ all\ nodes\ during\ t}$$

Using such a metric, we can recognize nodes that have experienced large weight increases relative to the rest of the nodes in a studied network during specific time periods. For example in the word network, we may notice that most of the publications in the software engineering field used the word "java" in their title during a specific year. Such a finding may indicate that research in the field during that year has focused on java technologies. We may notice in later years that such focus on java has shifted to a focus to a more general term ("distributed") due to the decline in the metric value for the word java and its increase for the word distributed. The interpretation of changes to the W metric varies between networks. Table 4 describe some possible interpretations for the changes in the W metric for each network type.

Neighbors Growth

We measure the increase of the number of neighbors of each node in the network during a time period. Using such a metric N, we can recognize nodes that have experienced large increases in interaction or collaboration with other nodes in a studied network. For example in the author network, we may notice that over time researchers are gaining new collaborators. The interpretation of changes to the N metric varies between networks. Table 5 describe some possible explanations for the changes in the N metric.

3 Evolution Spectrographs

In the previous section, we presented metrics to study interesting properties of networks. To study changes to these metrics in large networks, we use evolution spectrographs. An *evolution spectrograph* is analogous to sound spectrographs. A spectrograph is a color-coded evolution visualization technique which visually characterizes how a spectrum of com-

ponents change over time. A spectrograph can be tailored to examine a variety of aspects of historical data and to yield insights in understanding the process of evolution. By using a visual representations to study the historical data and its corresponding metrics, we are shifting work from the human cognitive system to the perceptual system and making use of humans' ability to detect patterns and anomalies, and to process large volumes of visual data quickly. We now describe evolution spectrographs by drawing an analogy to sound spectrographs.

Spectrograph Dimensions
A sound spectrograph provides a visual representation of the frequency content of sound and its variation in time. It is normally presented in the form of an XY graph, in which the horizontal axis X denotes the time dimension, the vertical axis Y denotes the frequency range, and the brightness of a position indicates the relative amplitude of the energy present for a given frequency and time. Analogous to sound, evolution of a network can be characterized in terms of *time*, *spectrum*, and *measurement*, and then visualized using spectrographs. We now provide our interpretations of these three dimensions.

Time
The time dimension denotes periods within the lifetime of a network. Time can be measured in two ways. We can measure time in units of evolution events, such as software releases. Or, we can use fixed-length periods as time steps, such as months and years. Depending on the purpose of our study, we need to measure time differently. For example, for an author network we can measure time in units of years since conferences occur yearly. For a subsystem network, we can measure time in units of software releases because the system structure likely undergoes substantial changes between releases. If we want to study developer activities (in a developer network), a measurement based on fixed-length periods ($e.g.$, month or quarter) may be more appropriate.

Spectrum
Analogous to sound decomposition into frequency components, a network is composed of nodes. These nodes provide a measurement basis for the Y axis. In the spectrum of sound, frequency components are arranged into an order according to their values. Similarly, network units (nodes) must be ordered by a particular property. In this paper, we order nodes by their creation time (appearance in the data set). This ordering technique permits us to visualize the growth curve of the network as part of the spectrograph. The growth curve is represented by the upper envelope of the spectorograph (see Figure 4). In Section 4, we present several concrete examples of spectrum.

Measurement
For a component in the spectrum, we can measure a particular aspect or property of that component at any points during the lifetime of a network. For our purposes we use the W and N metrics defined in the previous section.

Spectrograph Model
The spectrograph uses a matrix M as its underlying data model (see Figure 1). For a given spectrum, each of its components (c) will be measured according to a particular property (p). A row in the matrix stores a vector of values that represents the evolution history of a spectrum component. A column stores a snapshot of evolution states for all components in the spectrum at a particular time point or during a particular period. If the spectrum contains m components (c_0, c_1, ..) and time is measured using n discrete points (t_0, t_1, ..), the matrix will have the dimension of $m \times n$. We view such a matrix as a metrics-based representation that mathematically characterizes the history of a network.

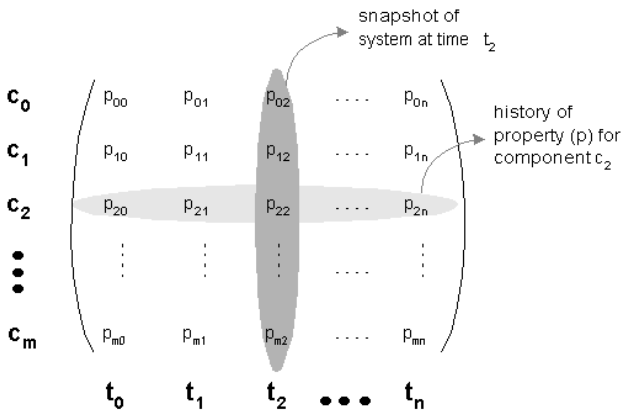

Figure 1: Spectrograph Model

Spectrograph Coloring
After we have computed an evolution matrix M, we use colors to code values stored in M in order to produce the final spectrograph. The coloring of the spectrograph permits us to easily distinguish patterns in the historical data. These patterns are then examined closely to gain a better understanding of the evolution of the studied network. This approach gives us a better view of large amount of data, in contrast to other graph based approaches which depend on aggregating the historical data into one or few data values for the whole system in a metric plot. These aggregation techniques are likely to hide some of the complex and interesting patterns that appear in rich historical data.

In previous studies, we found that the coloring must be specially tailored for various subjects and purposes [19]. We use a quartile coloring method for the case studies presented in this paper.

Quartile Coloring
This coloring method is based on the idea of box plots. By calculating the median and the quartiles (the lower quartile is the 25th percentile and the upper quartile is the 75th percentile), the value range of the studied metric is divided into

four quarters, which are associated with four different colors respectively. In our case studies, we have chosen red, yellow, light-green, and light-grey, as shown in Figure 2.

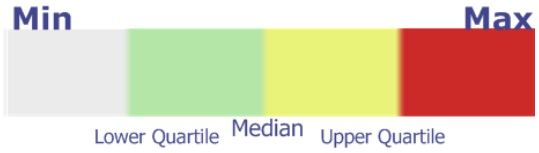

Figure 2: Quartile Coloring

4 Case Studies

Through the evolution spectrograph which uses a quartile coloring, we can closely study and easily recognize evolutionary patterns in the studied data. In this section, we show examples of several patterns that the spectrograph uncovered in each of the studied data sets and their corresponding networks.

Study 1: Publication Records – Word Network

For the word network, we studied the top one hundred most active words throughout the lifetime of the studied publication data. These words were in the titles of over seven thousand papers during a 25 year period. The studied words are chosen by summing up the W metric for each word for all 25 years then picking up the top 100 words. To clean up the studied data, we removed stopwords (such as "the" and "of") and used a Porter stemmer to derive the root of each word in the title of a paper (for example, truncating "extracting" to "extract" and "experience" to "experi") [15].

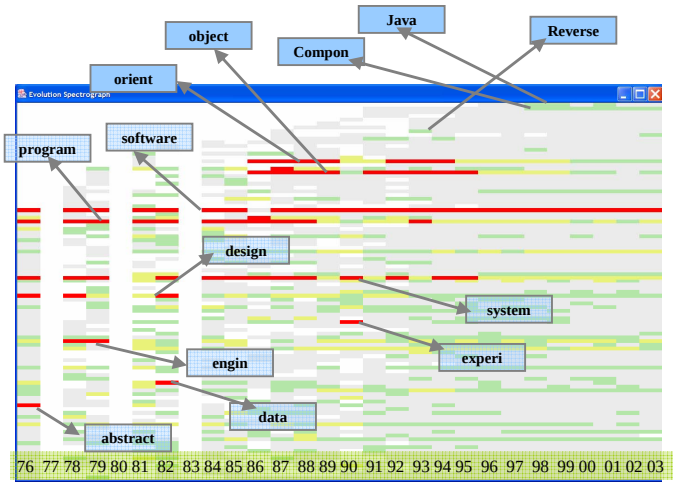

Figure 3: A Spectrograph for Words in Software Engineering Publications (Node Weight)

Figure 3 shows the evolution spectrograph for the node weight W for the top 100 nodes (words):

- A dark color (red) for a cell indicates that a word was in the title of a large number of publications during the corresponding year. For example, the word 'experi' in 1990. This may be an indication of the push in the software engineering area in the early 90s to engage practitioners in the research conducted by focusing on experience reports from industry.
- A dark colored horizontal line indicates that a word has been popular over a long period of time. For example, the words "software", "program" and "system". This finding is not surprising given that such terms are central to most research work related to software engineering.
- The spectrograph shows the slow growth of the usage of new terms. The latest popular terms are "java" and "compon".

Furthermore, we note a number of interesting patterns:

- The terms "object" and "orient" (corresponding to "oriented") have a high tendency to occur together in the same paper title. Nevertheless, the history of the term "object" traces further back in history in references to other usage such as "object code".
- The terms "java" and "compon" (corresponding to "component") have gained popularity in the recent years.
- The term "reverse" shows up 1993 which is the same year that the Working Conference on Reverse Engineering (WCRE) commenced.

Study 2: Publication Records – Author Network

We now look at another network generated from the publication records data: the author network. For the author network, we present two spectrographs: the node weight W and the neighbors growth N spectrographs. We again focus on the top one hundred most active authors in the field of software engineering out of almost ten thousand authors (*see* Table 2).

The node weight spectrograph, shown in Figure 4, reveals a rather interesting pattern. It shows that in the early years of the field of software engineering a small number of authors contributed a large percentage of the work in the field. This is visible in the dark (red) area in the lower left corner of the spectrograph in Figure 4. As the field evolved the contributions have become more varied and distributed across the researchers. The spectrograph also shows that the number of authors has been growing at a large rate. We believe that these findings are good signs since no single researcher or group of researchers are the main drivers of the field instead the field is evolving through the collaborative efforts of many of its members.

To examine the collaboration between authors in the field, we study the spectrograph for the neighbors growth for each node (shown in Figure 5). This spectrograph shows very light colors in the lower left corner of the graph while in the right side of the graph we see darker colors (green and yellow). This coloring pattern indicates that recently more

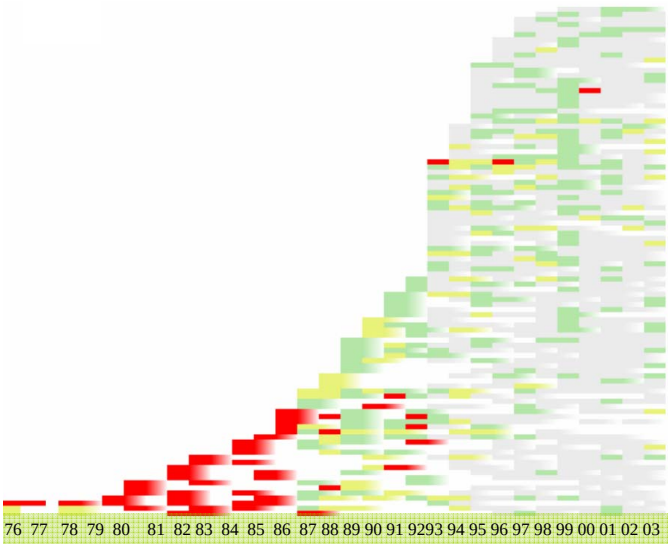

Figure 4: A Spectrograph for Authors in Software Engineering Publications (Node Weight)

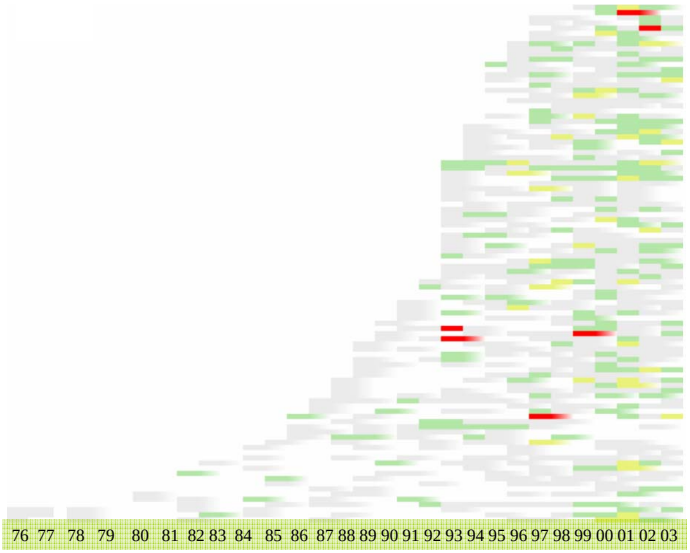

Figure 5: A Spectrograph for Authors in Software Engineering Publications (Neighbors Growth)

and more authors are collaborating with new authors. This is likely due to the growth of popularity and accessability of the Internet and Electronic email which represent great collaboration mediums for authors and researchers worldwide [9].

Study 3: Source Control Data – Subsystem Network
Encouraged by the patterns revealed by the spectrographs for the publication records data, we used spectrographs to study the source control data for several open source projects. We present some of these spectrographs and discuss the most interesting findings.

We examined the NetBSD source control data for the ten years, starting from March 1993. We divided the data into 30 time periods each time period is four months long.

NetBSD is an operating system derived from 4.4BSD and 386BSD. It is being developed with a primary focus on creating an extremely portable and flexible OS. It runs on over 30 hardware platforms and provides a lot of flexibility to enable research and experimentation with many different types of hardware, and protocols.

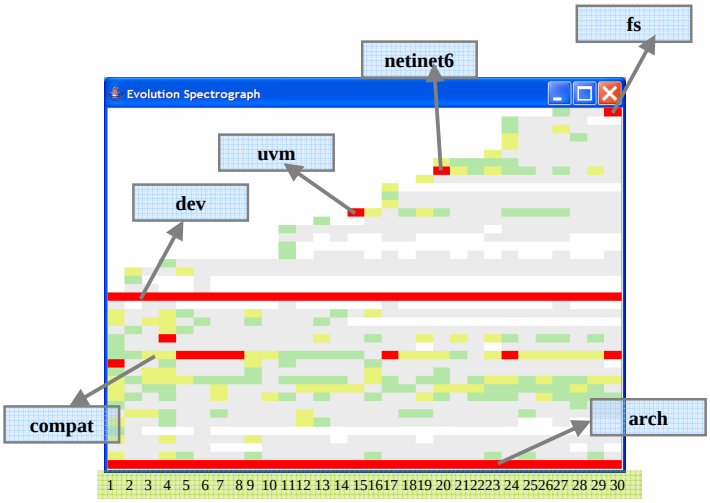

Figure 6: A Spectrograph for Top Level Subsystems in NetBSD (Node Weight)

Figure 6 shows a spectrograph of all 43 high level subsystems in NetBSD. We now discuss a few noteworthy patterns:

- The “arch”, “compat”, and “dev” subsystems have always been actively modified throughout the lifetime of the NetBSD project. The “compat” subsystem is responsible for ensuring compatibility to other operating systems such as Linux, Solaris, and Ultrix. The “dev” subsystem contains the code for the various device drivers. The “arch” subsystem contains the code needed to support the various hardware architectures such as x86 and Sparc. The continuous modifications to these three subsystems indicates that the project is actively adding and maintaining support for a variety of hardware architectures and devices (the main design goal of the project).
- The “fs” subsystem is a recently added subsystem (it appears in time period 27). This is a surprising finding since we would have expected NetBSD to support several filesystems throughout its lifetime. A closer examination of the source control data reveals that the appearance of the “fs” subsystem coincidences with a source code refactoring which grouped several of the high level filesystem subsystem, such as “msdosfs”, “ntfs” and “smbfs”, under the newly created “fs” subsystem.
- The “netinet6” subsystem appears in time period 20 (mid 1999). The “netinet6” subsystem supports the

IPv6 internet protocol, its appearance corresponds to the growth of awareness for the need for the IPv6 protocol to deal with the explosive growth of networked devices. The "netinet6" subsystem has been actively changed at a high rate. Its rate of change has been slowing down over time (going from red to grey). This may be considered as an indicator that it is becoming more stable and robust.

- The "uvm" subsystem appears in time period 15 (early 1998). It replaces NetBSD's virtual memory subsystem with a new system that is specifically designed to provide NetBSD's I/O and IPC subsystems with a range of flexible data movement mechanisms. Examining the "uvm" spectrum we can monitor its stabilization over time. We as well notice that the "uvm" went through another stage of active change between periods 24 to 27.

Studying Co-Change

As a software system evolves, it is preferable that working on features or fixing bugs would require localized changes. The need for a subsystem to be modified when another subsystem is modified is an indication of an implicit dependency that may introduce bugs in the future [7]. Therefore, a good design should aim to minimize the need for co-changing other subsystems. We are interested in monitoring growth of co-change dependencies between subsystems over time. By studying the neighbors growth over time, we can monitor the decay of the design of a software system.

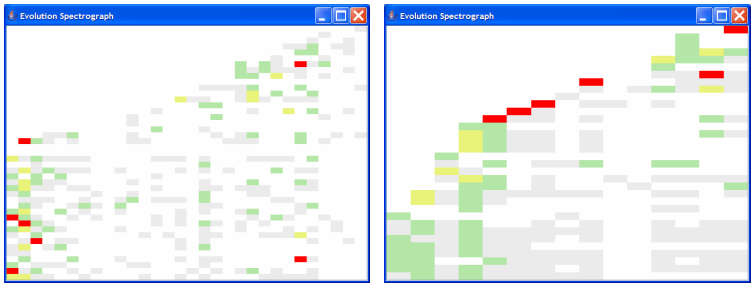

Figure 7: A Spectrograph for Top Level Subsystems for NetBSD and KOffice (Neighbors Growth)

Figure 7 compares the addition of new neighbors over time for two projects (NetBSD and KOffice). The *KOffice* productivity suite is written in C++. It is an integrated office suite for *KDE*, the K Desktop Environment. The full suite is developed by a community of software developers online under an open source license. It features a full set of applications: *KWord* a word processor, *KSpread* a spreadsheet application, *KPresenter* a presentation program, *Kivio* a visio-style flowcharting application, *Karbon14* a vector drawing application, *Krita* a raster-based image manipulation program like Adobe Photoshop, *Kugar* a business reports generating tool, *KChart* a chart drawing tool, $KFormula$ a powerful formula editor, and *Kexi* a small database similar to Microsoft Access. It has 34 high level subsystems most of these subsystems correspond to the applications offered in the suite.

Examining Figure 7 reveals that:

- The first period of both projects tends to have darker colors as more dependencies are created between subsystems.
- Throughout the lifetime of NetBSD, its subsystems have gained new neighbors ($i.e.$ other subsystems which had to be modified along with them). The same holds for KOffice. Nevertheless, for KOffice we notice that the growth has been larger over time (more grey areas). Furthermore, it seems that new subsystems in their first period (red diagonal in the KOffice spectrograph) have a tendency to interact with a large number of subsystems. We believe that this may be an indication that the current KOffice design does not enforce strict information hiding principles between its high level subsystems, since the initial code for each subsystem which is likely to suffer the least from design decay tends to interact with too many subsystems.

Study 4: Source Control Data – Developer Network

We examine the weight growth of each node (developer) in the developer network for the KOffice project using a spectrograph shown in Figure 8. The spectrograph shows that several of the original developers of KOffice are no longer modifying the code of the project (the white area in the lower right corner of the spectrograph), this may be due to them departing from the project or assuming managerial roles.

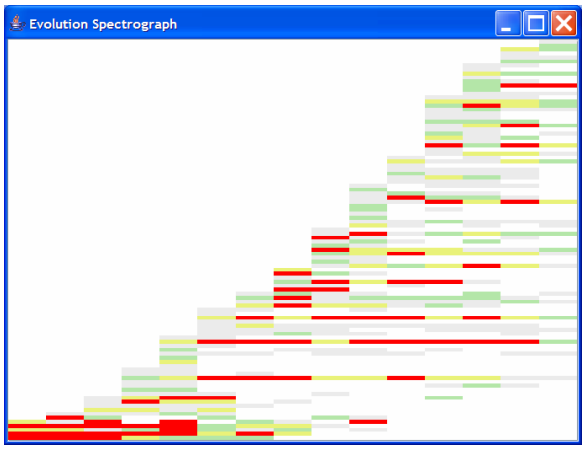

Figure 8: A Spectrograph for Developers in KOffice (Weight Growth)

Evolution spectrographs can visualize changes to a variety of metrics while minimizing the need to aggregate the data into a few metric values. The presented case studies used evolution spectrograph to analyze and discover patterns in large complex historical data.

5 Related Work

The paper presents techniques and metrics to visualize the evolution of complex processes. The idea of applying social network analysis techniques to the source control data has been proposed by Lopez-Fernandez *et al.* [12]. The authors propose measuring several graph metrics and use aggregation techniques (averaging and weighting) to visualize the evolution of large open source projects. Our approach makes use of spectrographs to reduce the need for aggregating the data into a single number since the spectrograph permits us to examine closely the evolution of several data entities at once.

The literature on software engineering is rich with studies about software evolution. Most of the studies focused on using simple numerical plots to gauge the evolution of a software system, few papers have proposed the adoption of specialized visualization techniques. Our work contributes to that venue of research. We now present and contrast our work to prior work along the same venue of specialized visualization techniques.

Gall and Jazayeri present a 2D or 3D visual representation for examining a system's release history [8]. They visualize system structure, historical evolution, and software properties simultaneously in one view. The system structure is displayed by 2D or 3D graphs. The third dimension is used to represent time. Colors are used to represent a particular property (e.g., version numbers, code size, etc.). They have applied this technique to find notable changes from the release history and to support the system's future evolution. In more recent work, Gall proposes a visualization technique specialized for detecting unintended coupling between modules [7]. A similar approach is proposed by Bieman *et al.* to visualize the change architecture of object oriented systems [1].

The Evolution Matrix is used to visualize the evolution of OO software systems [10]. Each class is represented as a box with the dimensions of the box determined by metrics. For example, the number of instance variables may determine the width and the number of methods may determine the height. The layout and shape are used to highlight change patterns over time. Lanza has applied this visualization technique on a number of OO software systems and identified several interesting evolutionary patterns about classes, which include Pulsar, Supernova, White Dwarf, Red Giant, Dayfly, Stagnant, and Persistent [10]. By contrast, the spectrograph relies on coloring to emphasize visual cues and scales the analysis to large historical data sets.

The SeeSoft view is used to visualize historical code change data stored in source control repositories [4]. Each line of code is reduced to a pixel in the SeeSoft view. Each pixel is colored based on historical attributes that are calculated for the corresponding line. Whereas the evolution matrix and the spectograph visualize the evolutionary process, the SeeSoft view condenses the rich evolutionary process into a single color for each pixel. Additionally, Eick *et al.* describe how several views such as bar-graphs, pie-charts, matrix views, and cityscape views can be applied to visualize a large number of statistics from many different perspectives [5]. Recently Froehlich and Dourish presented a tool called Augur with added information about the developers working in the project to the SeeSoft views. The Augur views condense the evolutionary process of a software entity into a single pixel [6].

GEVOL is a graph-based system for visualizing software evolution [2]. A sequence of graphs are created to depict the different states of a system at given points in time. GEVOL preserve the viewers mental map as it moves between graphs by using advanced force-directed layout algorithms. Colors are applied to indicate change over time. The spectrograph gives a static view of the evolution process and is able to scale to analyze large data sets easily.

Revision towers are used to visualize the change history stored in source control repositories [17]. A tower like view is created for each file. The view shows all the revisions of the file and the relationships between revisions and source releases. Towers corresponding to all files in the repository are displayed in a grid which fills the available display area. The towers are ordered according to the date of file creation. Although a revision tower provides very detailed information about the change history of a file, it does not scale for large software systems or other large data sets such as publication records. Moreover, the approach does not assist in highlighting interesting evolutionary patterns between studied entities such as files.

Rysselberghe and Demeyer present a 2D plotting visualization technique for recognizing relevant changes [16]. Applying this technique on the change history of Tomcat, they have been able to identify unstable components, coherent entities, design and architectural evolution, and fluctuations in team productivity. Their work does not explore varied coloring.

6 Conclusion

The paper presents two main ideas. First, it proposes the mapping of large historical data into a network with nodes and edges. This network can be studied using techniques that are independent from the data. The results can be interpreted back in context according to the studied data. Second, the paper demonstrates the use of spectrographs to support evolutionary studies. Instead of using aggregation techniques such as averaging, the data is studied at a higher granularity. This higher granularity permits the discovery of interesting patterns and events that are likely not to be visible as demonstrated by prior studies by Gall [7] and Godfrey [13].

Spectrographs are useful for highlighting trends and anomalies in the metrics for studied data entities. Researchers can then develop hypotheses that could be tested by examining closely the data to understand the reasons behind such

anomalies (red areas).

The case studies, presented in the paper, show several interesting aspects of the evolution of data. For publications in the field of software engineering, the main findings are:

- Collaboration between researchers has increased over time.
- Contributions to work in the field have become more varied and less driven by a selected few, instead a large number of researchers contribute to publications and research in the field.

For source control data, spectrograph have highlighted:

- The evolution of source code components as they stabilize.
- Large refactoring events such as the consolidation of all filesystem related subsystems under one higher level subsystem.

Spectrographs have revealed interesting evolutionary patterns in complex data sets such as the source control data for large software projects. Although in this paper we propose the creation of four types of networks, a number of other networks could be created and studied in hopes of revealing other useful patterns and events that could assist in recovering complex historical events. Moreover, other coloring techniques and metrics should be explored since they may reveal several interesting patterns.

Acknowledgments
The anonymous METRICS reviewers gave helpful comments on earlier revisions of this paper.

REFERENCES

[1] J. Bieman, A. Andrews, and H. Yang. Understanding Change-proneness in OO Software through Visualization. In *Proceedings of the International Workshop on Program Comprehension (IWPC 2003)*, pages 44–53, Portland, Oregon, USA, May 2003.

[2] C. Collberg, S. Kobourov, J. Nagra, J. Pitts, and K. Wampler. A system for graph-based visualization of the evolution of software. In *Proceedings of ACM Symposium on Software Visualization*, pages 77–86, San Diego, California, June 11-13 2003.

[3] DBLP Bibliography Home Page. Available online at `http://www.informatik.uni-trier.de/~ley/db/`.

[4] S. G. Eick, T. L. Graves, A. F. Karr, J. Marron, and A. Mockus. Does Code Decay? Assessing the Evidence from Change Management Data. *IEEE Trans on Software Engineering*, 27(1):1–12, 1990.

[5] S. G. Eick, P. Schuster, A. Mockus, T. L. Graves, and A. F. Karr. Visualizing software changes. *IEEE Transactions on Software Engineering*, 28(4):396–412, April 1002.

[6] J. Froehlich and P. Dourish. Unifying Artifacts and Activities in a Visual Tool for Distributed Software Development Teams. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 387–396, Edinburgh, UK, May 2004.

[7] H. Gall, M. Jazayeri, and J. Krajewski. Cvs release history data for detecting logical couplings. In *IEEE International Workshop on Principles of Software Evolution (IWPSE03)*, pages 13–23, Helsinki, Finland, Sept. 2003.

[8] H. Gall, M. Jazayeri, and C. Riva. Visualizing software release histories: The use of color and third dimension. In *Proceedings of the International Conference on Software Maintenance*, pages 99–108, Oxford, England, August 30-September 3 1999.

[9] A. E. Hassan and R. C. Holt. The Small World of Software Reverse Engineering. In *Proceedings of WCRE 2004: Working Conference on Reverse Engineering*, Delft, Netherlands, Nov. 2004.

[10] M. Lanza. The evolution matix: Recovering software evolution using software visualization techniques. In *Proceedings of the International Workshop on Principles of Software Evolution*, pages 37–42, Vienna, Austria, September 10-11 2001.

[11] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski. Metrics and laws of software evolution the nineties view. In *Fourth International Software Metrics Symposium (Metrics97)*, Albuquerque, NM, 1997.

[12] L. Lopez-Fernandez, G. Robles, and J. M. Gonzalez-Barahona. Applying Social Network Analysis to the Information in CVS Repositories. In *International Workshop on Mining Software Repositories*, Edinburgh, UK, May 2004.

[13] Michael W. Godfrey and Qiang Tu. Evolution in open source software: A case study. In *IEEE International Conference on Software Maintenance (ICSM 2000)*, pages 131–142, San Jose, California, Oct. 2000.

[14] D. E. Perry. Laws and principles of evolution. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, page 70, Montreal, Canada, Oct. 2002.

[15] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–9137, 1980.

[16] F. V. Rysselberghe and S. Demeyer. Studying software evolution information by visualizing the change history. In *Proceedings of the International Conference on Software Maintenance*, pages 328–337, Chicago, Illinois, September 11-14 2004.

[17] C. M. B. Taylor and M. Munro. Revision towers. In *Proceedings of the 1st International Workshop on Visualizing Software for Understanding and Analysis*, pages 43–50, Paris, France, June 26 2002.

[18] W. F. Tichy. RCS - a system for version control. *Software - Practice and Experience*, 15(7):637–654, 1985.

[19] J. Wu, A. E. Hassan, and R. C. Holt. Exploring Software Evolution Using Spectrographs. In *Proceedings of WCRE 2004: Working Conference on Reverse Engineering*, Delft, Netherlands, Nov. 2004.