Software Process Recovery

Abram Hindle
ahindle@swag.uwaterloo.ca
University of Waterloo
Canada

October 6, 2009

Abstract

When developers create software they leave a trail of artifacts in their wake. These artifacts include source code changes, discussions over electronic mediums, documentation, tests, build scripts, requirements, etc. We propose to investigate methods in which one can discover and recover the underlying processes that developers follow, by analyzing the artifacts they leave behind. We wish to see if their behaviour that relates to these artifacts correlates with underlying processes developers are following. We also wish to determine if developers are following the processes they claim to be following. Other researchers have approached this problem by tooling the process that the developers are following in order to record more information. By contrast, we want to recover development processes *ex post facto*, after the fact. We propose to use *process recovery* to expose, recover and validate the underlying behaviours and processes that developers follow, collaboratively, to build software. We propose an analysis framework to help research process recovery.

1 Introduction

When developers create software they often leave a record of their work. This record comes in the form of software development artifacts. These artifacts range from versions of the source code itself, to revisions to source code, to project documentation, to discussions about the project, and to other events and documents. These records, left behind, provide valuable historical trails which we hope will give us insight into the underlying processes used to develop this software.

The field of *mining software repositories* (MSR) [67] is dedicated to exploiting and understanding these artifacts of development and inferring the relationships between them. *Process recovery* is a sub-field of MSR but closely related to *process mining* [123] and *process discovery* [17, 19, 18], effectively combining MSR work with process work. What we propose is to further the state of the art of process recovery research and consolidate it with much of the existing mining software repositories research. The purpose of process recovery is to be able to retroactively recover process from project histories without having to rely on a previously heavily tooled process as suggested by Cook and Wolf [17, 19, 18]. Process recovery is the elicitation of underlying behaviours and informal and formal processes that developers follow while building and maintaining a software project, recovered from the artifacts that these developers leave behind. This paper and our proposed research will attempt process recovery from information left behind by developers and not rely on modifying the actual development at run-time.

Process recovery relies on the underlying artifacts to describe the development processes that have taken place. These processes and behaviours are documented within the project and recovered from artifacts such as source code, documentation, build and configuration management scripts, test scripts, mailing lists, bug trackers, and revisions to all of these artifacts. The behaviours discovered and the underlying processes recovered from these behaviours can be used to characterize the processes used to develop a software project. Some of the methods used for process recovery are already used to mine business processes.

Process Mining	Mining of business processes at run-time
Process Discovery	Mining of software processes at run-time
Process Recovery	Mining of software processes after the fact

Table 1: Definitions used throughout the rest of the paper

Process mining in the context of business processes, processes which define business tasks, has been heavily investigated by Van der Aalst et al. [123]. Process mining is the run-time investigation of and recovery of business processes. With respect to software engineering, and mining software for processes, Cook et al. [17, 18, 17] attempted to discover underlying processes, referred to as process discovery, by recording developer actions at particular times. Cook took an existing project being developed, and inserted measurement devices into the process in order to record process-related information, as the project was being developed. In contrast, process recovery does not require tooling an existing process, it involves mining artifacts left behind for evidence of underlying processes. In this particular case we respect that some projects have documented their processes but we hope that we can help validate that these processes are being executed by recovering evidence of their use from the artifacts left behind. Thus process recovery seeks to expose and uncover processes after the fact, or in an *ex post facto* fashion.

The usefulness of process recovery lies mainly in reporting and validating existing knowledge about a particular project. Process recovery could be used to analyze past behaviours and processes within a software project, and determine what methods of development took place. One could produce reports of the observable and recorded behaviours that occurred within an iteration and how close these behaviours follow existing development processes and project proscribed development processes. Process recovery would give some semblance of what actions developers were taking as well was what work was done in the past. Process recovery could also aid in extracting and describing the processes and behaviours that occurred within successful projects. These processes might be useful for planning new projects. Alternatively the unsuccessful processes can be mined; perhaps there is a correlation between unsuccessful projects and the processes they use. Thus process recovery can mine processes, and potentially determine what the underlying observable process is.

Users of process recovery would be those stakeholders who are interested in how the development of the project was executed. Potential users of process recovery are developers, managers, consultants, and those responsible for acquisitions. Developers could use process recovery to review how their coworkers were developing their current software project. Managers could verify what processes their developer's were following. Consultants could determine the effort and quality of processes used within a project, they could then determine how much work it took to implement the project. Process recovery is useful for *process verification*. Managers could investigate the development artifacts of outsourced components and try to validate if the outsourced team was following the processes they had agreed to follow. Process recovery can provide stakeholders with evidence based suggestions about what the underlying and observable processes in a project are.

Our goal will be to integrate this research in order to recover underlying software processes that are observable from the artifacts that developers leave behind. The rest of this paper will include:

- A survey of the data analysis techniques, process, mining software repositories and process recovery research (Section 2).
- A proposal for research into process recovery via integrating current and previous efforts (Section 3).

Next we will provide a survey of the groundwork for software process recovery. Afterwards we propose a line of research within process recovery which hinges primarily on the integration of already existing techniques and the refinement of these techniques into a relatively holistic framework for process recovery.

2 Previous Work

Our research will focus on automatic and semi-automatic process recovery. There is much previous relevant work related to this goal of semi-automated process recovery. This work is built up from four main areas: *process*, the formal and adhoc steps behind development; *data analysis*, the tools that we can rely on when we extract data from artifacts; *mining software repositories*, the field and the associated methods and tools that are focused on extracting information from artifacts created during development; *software process recovery*, how previous authors have attempted software process recovery.

2.1 Stochastic Processes, Business Processes and Software Development

Development processes, or simply processes, guide the development of software, whether they be informal and adhoc, or formal and regimented. Process is a very general word which has multiple meanings depending on the field. In this document, we will use the term process to refer to three related concepts: software development processes, business processes, and stochastic processes. We describe each kind of process in the following subsections. Software process recovery attempts to model these processes from the evidence left behind by developers.

2.1.1 Stochastic Processes

Stochastic processes [96] are processes with possible probabilistic variation, that is sequences of behaviour which are often modelled probabilistically. These are much lower level than software development processes, but often represent the statistical properties of a data-set. Stochastic processes can be described by parametrized probability distributions, such as the Poisson distribution or the Pareto distribution [48], since they involve randomness and error. The processes could be simple relationships between inputs such as time or effort and other measured values like growth. With respect to software development these processes are sometimes the result of modelling the growth of a software project.

Growth models of software, and some models of software evolution are stochastic processes as well. Lehman [76, 75, 78, 77] described laws of software evolution relating to life cycle processes. In general Lehman's laws suggest that growth cannot persist in the long-term without control because complexity due to growth will increase and slow down development. This was confirmed by Turksi et al. [121], who found that on some large systems growth was sub-linear. Counter examples arose subsequent in studies by Tu et al. [36] who showed that Linux in particular had super-linear growth and did not follow the laws of evolution as laid out by Lehman. Herraiz et al. [48] mined many software project and did growth modelling with various software metrics and found that the distribution of the metrics used to measure growth usually followed Pareto distributions. This knowledge of underlying distributions allows us to expect certain behaviours when we measure and include these behaviours into our models.

Stochastic processes are useful as they help model underlying data, allowing us to predict or reason more effectively about what will be observed. Some processes can be modelled by well known distributions, which we discuss in Section 2.2. Stochastic processes are relevant to software process recovery because they are used to model the underlying data that was mined from the various artifacts that developers left behind. Stochastic processes are usually too low-level, to close to direct measurements, they need to be abstracted and aggregated into business processes and software development processes to be useful to software process recovery.

2.1.2 Business Processes

Business processes are sequences of related tasks, activities, and methods combined, but often within the context of a business operation. These processes can be executed by actors, such as people or software, with or without the help of software. Generally business processes describe a kind of collaboration of tasks that occur between actors. Van der Aalst et al. [123] describes *business process mining* as the extraction of business processes at run-time from actual business activities. Example business processes are: handling

customer returns at a retail store, authenticating clients over the phone, and creating a client record. Business processes can be fine-grained, e.g. dealing with tasks which allow for this. Van der Aalst deferred to Cook and Wolf when it came to process mining on software projects. Software development is a kind of information work, like research, and thus is not easy to model so formally and so fine grained. While process is still important to software, the business processes described by frameworks like *BPEL* [122] (business process elicitation language) do not model software development well. Business processes are related to software process recovery because tools such as sequence mining [68] can produce processes that are quite similar.

2.1.3 Software Development Processes

Software engineering makes use of software development processes. These are processes modelled after industrial work processes, such as the assembly line. Some researchers and practitioners hoped that software development processes would enable teams to produce software in a repeatable manner, akin to how many other engineered products were produced and designed. Software development processes are software development methodologies. The tasks in a software development process can include just about everything: requirement elicitation, design, customer interaction, implementation, testing, configuration management, bug tracking, etc. Some processes deal with a part of software development such as maintenance [74]. Software development processes are seen as a way to try to control the evolution of a software project in order to ensure that requirements are met and the project is successful.

The software development life-cycle (SDLC) [114] describes how software is often built, maintained, supported and managed. Most software development processes relate to some if not all of the various aspects of the SDLC. Software development processes relate to methodology related to software development, such as the waterfall model [109] and the spiral model [12]. More recent software development processes include the Unified Process [65] (see Figure 7 for a diagram of the Unified Process workflows over time), Extreme Programming (XP) [4], SCRUM [110], and many methodologies related to Agile development [49]. Most of the recent develop processes focus on smaller iterations, such that design and requirements can be updated as they become more clear with each iteration. Software development processes and life cycles seek to manage the creation and maintenance of software.

Meta-processes, such as the Capability Maturity Model (CMM) [99, 100], attempt to model the processes used to create software, much like ISO standard 9000 [117] attempts to model and document how processes are executed, tracked, modelled and documented. The CMM tries to model, track, and rank software process adherence.

Software development models from literature are typically high level, they do not have many tiny fine-grained sub-tasks, but companies often have much more detailed home-brew processes. There are some processes which can be modelled in a fine grained manner though, for instance researchers have modelled how bug reports are created and handled with in a bug tracker system. Their end process was essentially a state diagram of bug states [107]. Software development processes are what software process recovery tries to recover from the evidence left behind by developers. Thus software development processes model software development at a relatively high level when contrasted with finer-grained processes such as business processes.

2.1.4 Summary

We have covered three kinds of processes each increasing in their level of granularity and generality. At the intermediate level we have business processes which describe fine grained actions and tasks taken to fulfil a goal. Then at the high level we have software development processes which group together different efforts into a general process. We seek to extract all three of these kinds of processes from existing artifacts in an effort to recover the underlying processes that were executed to build the software project.

2.2 Data Analysis

In order to automate process recovery we will need to extract and process the data from the software development world. This data comes in many forms, from many repositories, as described by the mining software repositories community [67]. The main tools we use and describe include: statistical analysis, time-series analysis, machine learning, sequence mining, natural language processing, and social network analysis.

2.2.1 Statistics

Statistics [96] help us describe data and model data, and they are an integral part of data mining. In the rest of this section, we will summarize some commonly used statistical methods; the reader may choose to skip over this. The main statistical tools we use are distributions and summary statistics. Distributions are characterizations of how values, such as measurements, are spread out across their possible range. Distributions show the relative frequency of values occurring or having occurred within a data-set. Some distributions have been modelled as functions or shapes. These distributions are parametrized, so we can generate them from a function. Some common distributions include: the normal distribution (bell curve), exponential distribution, Poisson distribution, Pareto distribution, power-law distribution, Zipf distributions, etc. Distributions can be summarized or characterized by summary statistics. Summary statistics are metrics that describe aspects of a distribution such as the average, the median, the variance, skew, kurtosis, etc. These summary statistics can be used to describe the shape and expected range of values of a distribution without the need to show the distribution itself. Distributions and summary statistics tell us about the underlying data and help us model the data. In order to better model the data it is useful to see how similar distributions or models are to each other.

The statistical tools we use for comparing distributions include: the t-test, the χ^2 test, KolmogorovSmirnov test, the Lilifors test and Euclidean distance. These distribution similarity tests can be used to indicate if underlying distributions are similar or if a distribution matches a well known distribution like a exponential distribution, a Poisson distribution, a power law distribution, etc. The Lilifors and Kolmogorov-Smirnov tests are especially useful for dealing with software-related data because they are non-parametric, which means that they can be used to compare data distributions that are not Gaussian (normal); this can be contrasted with the t-test and chi^2 tests, which can be sensitive to other kinds of distributions. Lilifors and Kolmogorov-Smirnov tests are based on measuring the largest distance between two cumulative distribution functions (the integral the probabilistic distribution function). These tests allow us to compare distributions to each other, but sometimes we want to search for distributions that are similar to compare models to the underlying data.

One common way to compare models is to use linear regression. Linear regression tries to find the best fitting line through our data,; this is accomplished by trying to minimize the R^2 values, which indicate how much unexplained variance there is in our data compared to our best matching linear model [59]. These statistical methods allow us to compare and reason about data-sets, create model of the data-sets, and validate these models against the data-sets.

2.2.2 Time-series analysis

A time-series is a set of data points plotted over time. MSR research often deals with time-series, and much of the data collected from software repositories is data that occurs across time, which means the data has temporal components. The underlying data could be an event, it could be an aggregate of events, it could be a measurement at a certain time. Hindle et al. [57] discussed spectral analysis applied to time-series. Spectral analysis, using Fourier transforms, allows for underlying periodicities to be recovered from signals such as changes per day. This spectral analysis is similar to the *auto-correlation* used in time-series analysis [47]. Auto-correlation attempts to look for underlying cyclic behaviour by comparing a signal to itself. Much of the analyzed data from repositories is time-series data, utilizing these tools allows us to deal with time and look for potentially repeating patterns across time.

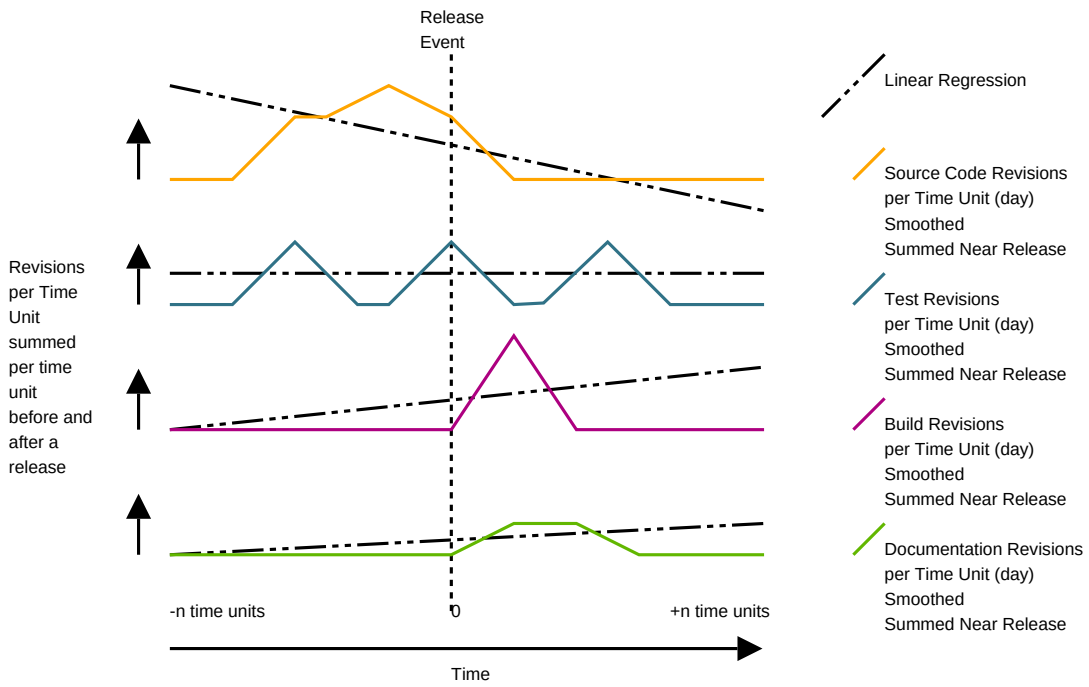


Figure 1: A project's change history split into four streams: source changes, test code changes, build system changes, documentation changes.

Time-series analysis relates to *longitudinal studies* [111]. These are studies of data and measurements over time. Longitudinal studies can employ multilevel modelling, which is a mixture of statistics and time-series analysis. Multilevel modelling first requires that we model measurements of entities (e.g., changes to a file, changes made by an author) over time. These are our first-level models. Then we can aggregate the parameters of these models and produce a second-level model which generally describes all of the underlying entities and the variances of each parameter. A common scenario for longitudinal data analysis is to model the performance of children in a classroom. The first-level models are those of the children and their performance throughout the school year. The second level model is the general model of the children altogether, the average rate of change of performance and the variance within the children's models. Longitudinal studies and multilevel modelling allow us to reason about general events like the major releases of a project, but also about the instances such as a single major release of a project.

We have used a rudimentary form of multilevel modelling [58] to describe how individual releases behave compared to the general trend of release behaviour within a software project. Figure 1 gives an example of how we analyzed behaviour around release time. Thus we have already utilized time-series data, Fourier analysis, and multilevel modelling in our own research [58, 57]. All three of these technologies are valuable for process recovery because they allow us to describe patterns that occur over time, find periodic behaviours, and model individuals and groups of entities which should enable us to determine stochastic processes and perhaps identify business processes by mining repeating behaviours.

2.2.3 Machine Learning and Sequence Mining

Machine learning [64] provides a powerful set of tools that helps process recovery by enabling automatic classification of entities based on historical facts. Machine learning is attractive because it can often utilize training data (the past) to classify the new data. It also enables bootstrapping, where you invest some work classifying changes manually and then delegate the rest of the work to the machine learner. Machine learning is a broad topic and we have used many machine learners to analyze and classify the maintenance purposes

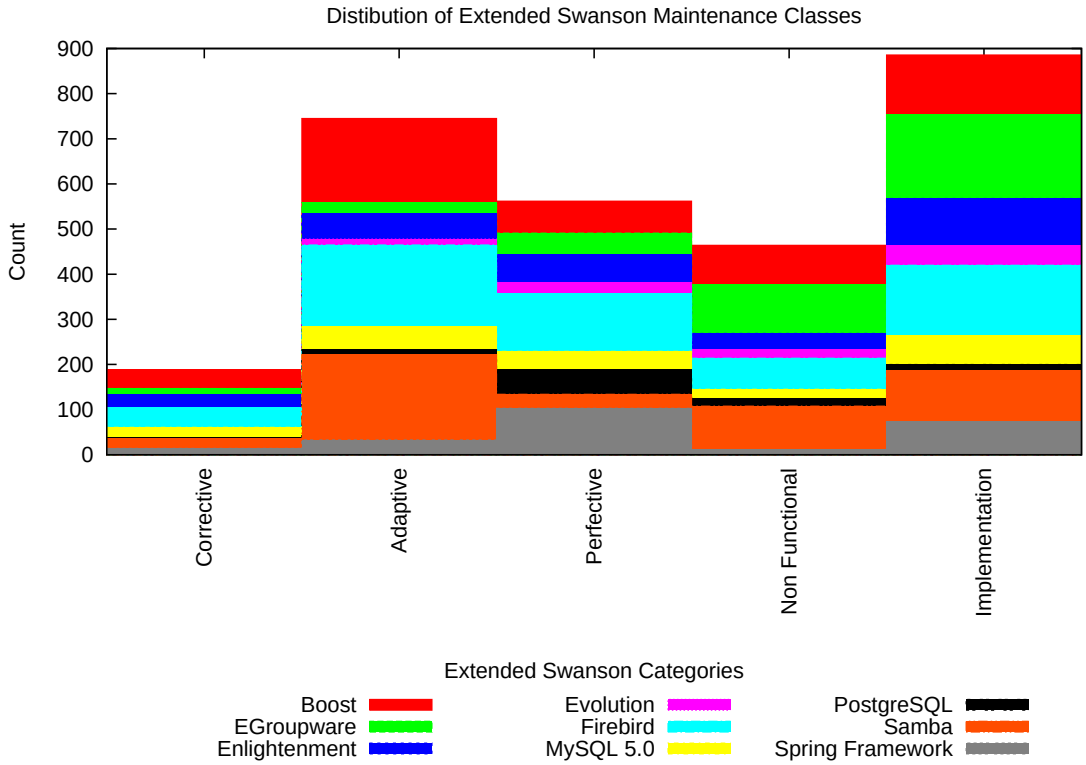


Figure 2: Proportion of large commits per project per Swanson Maintenance Category

of large revisions [52] (see Figure 2 for an example of distribution of large changes in multiple open-source projects).

For instance, sequence mining [68] attempts to find repeating sequences in streams of discrete entities. Sequence mining is valuable in process recovery because it can be used to find common chains of actions or events that occur within a project. Sequence mining can be used to mine processes at different levels of aggregation from fine grained events to the common sequences of phases of an iteration. Related to sequence mining is item-set partitioning [112], where similar behaviours are grouped along time in order to partition a time-line by observed self-similar behaviours. Thus machine learning, sequence mining and item-set partitioning help us infer new facts that we had not yet expressed. Machine learning can be used to help identify phases, while sequence mining can be used to find business processes inside of stochastic processes, and item-set partitioning can help identify when discontinuities occur thus perhaps helping us discover the end of a phase or an iteration.

2.2.4 Natural Language Processing

We often have to analyze text and source code. One method is to leverage techniques used in the field of Natural Language Processing (NLP) [69]. The most common NLP technique to analyze text is to abstract it as a word distribution. A word distribution is a word count, with optional stop word removal. Word counting produces a word distribution per each entity. These word distributions allow various kinds of processing, whether they act as input to a machine learner, or are used to represent entities in topic analysis.

Natural language processing (NLP) [69, 115, 50] deals with processing language in the form of speech or text into a computer-based representation of that speech or text. This processing might be used to better understand the underlying message, or it can be used to discriminate between low-level natural language

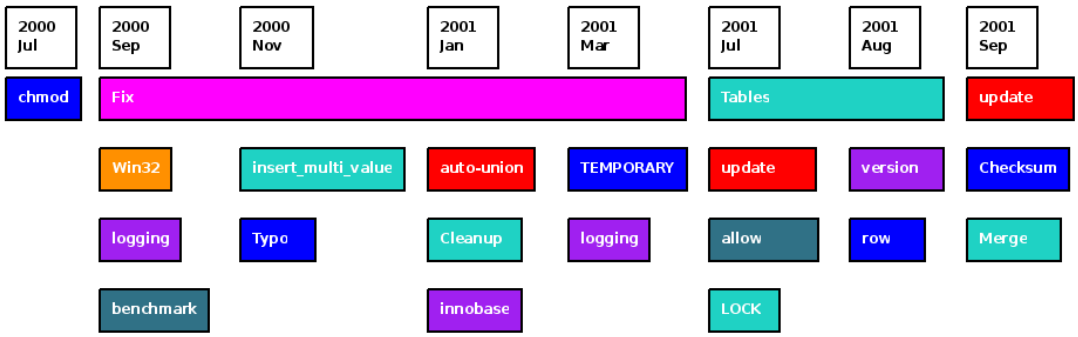


Figure 3: A manual example of topic analysis applied to MySQL 3.23 revisions. [60]

entities like words in a sentence or discriminate between high level entities like documents and sets of documents. For the purposes of this research we utilize NLP tools and methods in order to relate and process text messages left behind by developers. We also use similar structures and tools to process source code as well.

The basic NLP tools that we use are word distributions, stemming, and stop word filtering. Word distributions are a method of abstracting a block of text into counts of words, which can be further normalized by the size of the message. Word distributions are useful because one can apply a distance metric such a Euclidean distance, or the difference in CDF (Kolgoromov Smirnov test) in order to compare two messages. Stemming transforms words into their root forms. For instance a gerund like *biking* could be reduced to *bike*, and *dragged* to *drag*. Stemming cleans up tenses and modifiers to words, in an effort to make word distributions smaller and more similar if they cover similar concepts. Stop word filtering is another technique which removes words that are not important to the current analysis. For instance stop words, that could be removed when we are mining for concepts or modules, could be definite articles like *the*, other modifiers, or language keywords like *else*. Word distributions also serve as inputs for other NLP related techniques like topic analysis using LDA [11], LSI [87, 102], or ICA [22, 37].

Topic analysis or concept analysis is the automatic discovery and mining of topics (word distributions) that pervade a set of messages. For instance, a newspaper is often organized into sections by overall topic, and if a topic analysis algorithm were applied to all articles within a newspaper, we might expect that it might suggest topics that matched the main sections of a newspaper, such as life, entertainment, international, local, national, etc. Latent Dirichlet Allocation (LDA) [11] is an unsupervised topic analysis tool that is popular within the mining software repositories community [116, 81, 82, 80]. Other similar topic analysis techniques include latent semantic indexing (LSI) [87, 103], independent component analysis (ICA) [37], principle component analysis (PCA) [22], probabilistic Latent Semantic Indexing (pLSI) [63], semantic clustering [72, 73], etc. Sometimes source code and bug reports often serve as input to topic analysis tools, where as our own work used LDA to mine topics from commit comments [60] (an example of extracted topics appears in Figures 3 and 4).

NLP techniques are useful to process recovery because they recover information and associations from unstructured text data or data that is being treated as unstructured. These associations allow us to associate artifacts which different kinds of topics and potentially work-flows such as requirements, design, testing, or bug fixing.

2.2.5 Social Network Analysis

Social network analysis [124] is the structural analysis of social interactions between people or actors. A social network models the interactions between people and entities as a graph of entities as nodes and their relationships as arcs. These graphs and their interactions between nodes are measured using a wide variety of

Figure 4: Automatic topic analysis applied to MaxDB 7.500. This compact-trend-view shows topics per month for MaxDB 7.500. The x-axis is time in months, the y-axis is used to stack topics occurring at the same time. Trends that are continuous are plotted as continuous blocks. Trends with more than one topic are coloured the same unique colour, while topics that do not reoccur are coloured grey. The top 10 words in the topics are joined and embedded in box representing the topics. No stop words were removed. [60]

graph metrics such as centrality, connectedness, whether or not a node bridges two clusters, etc. The mining of social networks from software repositories is a popular field of research, whether it be social networks extracted from emails [7, 10] or other source code related artifacts [125, 9]. Social network analysis allows to study the structure of interaction, thus it can aid in process recovery by elucidating the relationships between developers, which might help us associate developers with project roles.

2.2.6 Summary

In this section we provided an overview of the tools that can be used to recover processes from artifacts left behind by developers. Statistics aids in modelling underlying data so we can describe the data and reason about it. Time-series analysis lets us reason about data that has temporal qualities. Machine learning and sequence mining allows software tools to make decisions and extract new facts based on the past. Natural language processing helps us deal with the large amount of unstructured and semi-structured natural language artifacts (such as messages or bug reports) that developers and users leave behind. Social network analysis lets us see the social interactions between actors and entities, allowing us to discover developer roles. All of these data analysis techniques have been leveraged in the field of mining software repositories and thus are quite relevant to process recovery.

2.3 Mining Software Repositories

Software process recovery relies on software related artifacts left behind by developers. These artifacts often appear in software repositories. Mining software repositories [67, 70] refers to the investigation and mining of data within software repositories such as source control, mailing lists, bug trackers, wikis, etc. Kagdi et al. [67] provide a survey and taxonomy of approaches and studies that exist within the mining software repositories (MSR) community [44]. Work in this area seeks to answer questions such as: which changes induce bugs or fixes [113], what information do software repositories tell us, how could new repositories be more useful, and what are the underlying properties of software, software evolution and changes to software. Process recovery is a sub-field of MSR because it relies on the repositories and the mining techniques that MSR literature employs in order to reason about underlying processes.

Much of the software and the software repositories that are mined in MSR literature is *Open Source Software* (OSS), this is due to the availability of OSS. Since OSS development artifacts are typically freely available, we can have repeated studies on the same corpus, thus allowing researchers to validate their results and the results of others, and to compare different approaches easily. Some researchers have studied and measured the general characteristics of Open Source Projects [14]. Mockus et al. [94] described the underlying methodology of OSS projects such as Apache and Mozilla. Much of the information they used was gleaned by mining the source control repositories of these projects. Thus much MSR research is executed on OSS.

2.3.1 Fact extraction

When one approaches a software repository often one has to reason about the data inside. An important first step is to extract this data from the repository. This step is called fact extraction. Many researchers have covered various details of how to extract facts from a SCS and store them into a database to be queried later [83, 34, 32, 27, 126, 45, 128]. Some issues that fact extracting handles: data cleansing, tracking identities, grouping revisions in commits, resolving symbols, etc. Fact extraction allows us to export a fact-base from a repository so we can reason about it further in other contexts. Fact extraction is necessary for process recovery because one needs to build up a fact-base in order to reason about underlying processes.

2.3.2 Prediction

We can use these extracted facts to help predict the future. Some developers and managers seek to predict what will happen next. Much work has been done on prediction by Girba et al. [35], where they attempt to predict effort much like how weather is predicted, based on past information. Others such as Hassan [46]

attempt to model change propagation in software systems via measures such as co-change. Herraiz et al. [48] attempted to correlated various metrics with lines of code (LOC) and growth, providing a power-law-like distribution which could be could used to predict metric values. One potential use of MSR research is to use past data to predict the future in order to aid project planning.

2.3.3 Metrics

In order to describe observations quantitatively we need to count and measure them, software metrics are meant to allow us to count and measure properties of software systems and their artifacts. Some metrics are related to process and deal specifically with measuring how many requirements a team has fulfilled so far [43]. Some metrics do not relate to the code itself, but about relationships that are observed external to the code.

Software Metrics — classic software metrics [25] include lines of code (LOC) [108, 48], and complexity measures such as Halstead's complexity [41] and McCabe's cyclomatic complexity [88]. There are object oriented (OO) metrics [104, 5, 42] like fan-in and fan-out, measurements of class hierarchies, calls-in and calls-out. Some have tried to correlate various software metrics with maintainability [15, 98] and have produced the maintainability index which supposedly indicates how maintainable software is. These classic software metrics describe underlying code but often for MSR related work they need to be extended by time.

Evolution Metrics — there are MSR and evolution related metrics [75, 90, 91, 89] that often deal with change or comparing one version of a system to the next. Yet there are other metrics that focus specifically on the change. Measurement of changes, whether they be revisions, diffs, deltas, structural deltas has been investigated in work by Ball et. al [3], Mens et al. [90], Draheim [23], German et al. [33], and Hindle et al. [56, 55]. Herraiz et al. [48] studied how metrics related to LOC over time. Hindle et al. [59] investigated the indentation of source code, particularly source code that appears in revision diffs, they found that the variance of indentation in a diff correlated with classic complexity measurements such as McCabe's complexity and Halstead's complexity. An example of how indentation metrics are measured is shown in Figure 5. Thus many MSR related metrics measure change itself, but there exists other metrics which measure temporal relationships or couplings between entities.

Coupling Metrics — some metrics are concerned with historical and logical coupling or co-change [29, 28, 46] between entities. Co-change is where two files or entities change together, and how often they change together. Co-change is also referred to as logical coupling [29, 28]. Co-change and coupling indicate cross correlations between entities. Zaidman et al. [127] studied the co-change between entities and tests, while Hindle et al. [58, 54] studied the co-changes between changes to source code, test code, build systems and documentation at release times. Coupling is useful within process recovery because if one wants to characterize a behaviour, one can use measures of co-changes across different classes of entities during a period.

2.3.4 Querying Repositories

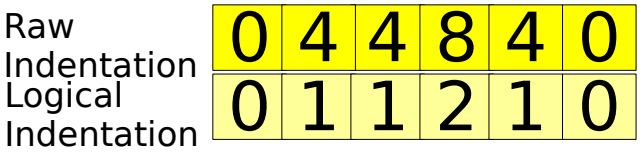
Once the facts have been extracted and the metrics suites run, we are left with a lot of information. There are many things an end user might want to do with this information: they might want to see related artifacts, they might want to query for structural qualities of changes, they might want to have changes described in a high level manner. If an analysis program can be rewritten as a query it saves both the researcher and the end user time.

A query system for developers to find hints to related documents was designed by Cubranic et al. [20]. Hippikat [20] is a SCS query system which attempts to link multiple "software trails" from different sources into one search engine for developers. Hippikat is much like a search engine rather than say a relational database.

Get the Diff

```
> void square( int * arr, int n ) {
> ····int i = 0;
> ····for ( i = 0 ; i < n ; i++ ) {
> ············arr[ i ] *= arr[ i ];
> ····}
> }
```

Measure the Indentation



Produce Summary Statistics

Metric	Raw	Logical
LOC	6.000	6.000
AVG	3.330	0.833
MED	4.000	1.000
STD	2.750	0.687
VAR	9.070	0.567
SUM	20.000	5.000
MCC	2.000	2.000
HVOL	152.000	152.000
HDIFF	15.000	15.000
HEFFORT	2127.000	2127.000

Figure 5: How indentation metrics are extracted and measured.

The SOUL system [13] allows for querying programs by structure and example, over time, but also poses queries in a Prolog like language, giving the query a lot of flexibility. This allows entities to be queried in a relatively unconventional manner. Related to SOUL is Semmle, SemmleCode and .QL, the semmle query language [79], together these tools provide an object-oriented query system to staticly analyze code. Semmle is used to query the underlying source code of a project by utilizing static analysis. There are also other logic-based query systems related to MSR.

Hindle et al. [51] created and defined a query language, Source Control Query Language, for querying data from SCS using first order and temporal logic queries. These queries were created to check for invariants within a project, as well as generally query a repository for change patterns. Hindle et al. built a system which uses first order and temporal logic to find entities, Others have written systems which take existing data and then use logic to describe the change.

Kim et al. [71] have studied how to described fine grained changes succinctly. They used a descriptive grammar of first-order logic and automatic inferencing in order to make a concise and small description of a change, which described whether it included certain files or if it modified certain structures within a program. Descriptions could be similar to: "All subclasses of Mapper were changed except for AbstractMapper". These concise queries work well when designing reports for end users who want high-level overviews.

Querying and inference are valuable tools to look for certain behaviours within a software repository. Inference allows for concise descriptions of changes while querying enables for concise extraction of patterns and results. Conciseness might be important to process recovery when used in the context of creating reports.

2.3.5 Statistics and Time-series analysis

Some MSR research seeks to quantitatively reflect and describe what is going on within a software repository. One way to do this is to utilize summary statistics, software metrics, and time-series analysis. Because so many repositories have entities with temporal aspects, time-series analysis and the extraction of temporal patterns is quite useful.

Large distribution studies have been executed by Herraiz et al. [48], Capiluppi et al. [14], and Mockus et al. [93]. In these studies metrics and summary statistics were used on numerous OSS projects. These studies provided more of a static model of what to expect from software measurements and were not so much about evolution.

With respect to time, much work is done on time-series and MSR related data. Israel Herraiz et al. [47] worked on ARIMA models of the number of changes. ARIMA models are meant to model and predict time-series data using techniques like auto-correlation. Herraiz studied many OSS projects and tried to characterize the ranges of the ARIMA model's parameters that could model these projects. Related work in mining recurring or repeating behaviour was done by Antoniol et al. [1], where they attempted to mine time variant information from software repositories using linear prediction coding (LPC). LPC is often used in speech audio compression. Hindle et al. [57] described how to pose these time-series and recurrent behaviour problems in the context of Fourier analysis, that is to treat time-series as signals and apply signal analysis techniques to the data itself. Time-series analysis helps us reason about changes and behaviour over time but if we have many entities with many time-series we can use tools like multilevel modelling.

Release Pattern Discovery [58, 54] is much like multilevel modelling [111]. Each release has its own parameters for its models (a linear regression or a simple slope of changes across an event). Release pattern discovery attempted to communicate the behaviour of developers at release time by breaking up fine grained changes, recorded in SCSs, into four streams: source code changes, test code changes, build changes, documentation changes. This enables the end user to describe a behaviour around a release in terms like, "source code changes general increased across the release, while testing changes were constant. There were usually no documentation before a release but many documentation changes after a release". This multilevel modelling allowed us to talk about individual releases as well as all releases. Software process recovery requires time-series analysis in order to reason about the fine-grained aspects of the various artifacts being analyzed.

2.3.6 Visualization

Visualization, with respect to mining software repositories, often attempts to reconcile the added complexity of time with already complex data. For instance when MSR and UML diagrams are combined you can potentially have a new UML diagram per each revision to the source code. Visualization seeks to deal with this complexity by leveraging visual intuition and reasoning to aid the understanding of the underlying data.

Much software visualization research is related the visualization of software design and architecture. Usually architecture is represented as graphs, where modules are nodes and relationships between modules are arcs. Sometimes hierarchical containment relationships are used to deal with tree-like data. Rigi [95], Shrimp [118] and LS-Edit [119] are interactive graph drawers which typical display hierarchical relationships of a software's architecture using both containment and arcs. Rather than architecture, some researchers have attempted to visualize the social structures within a project [97]. Most of these tools provide a static, although interactive view. These tools usually do not support visualizing graphs that evolve over time.

One way to visually model time and evolution is by animation. Visualizations that use 3D commonly use animation for navigation. Gall et al. compared releases via 3D visualizations [30]. Marcus et al. applied 3D visualization to source code [85]. While others will use animation to show the progress of time, for instance researchers have used VRML to animate software evolution matrices [92]. Beyer et al. [6] used animation and software evolution metrics to produce storyboards of changes to files. Other kinds of non-3D visualizations typically use graphs.

Much visualization has been applied to graphs and temporal-graphs, ranging from D'Ambros's radar plots [21], which indicated recency with distance, to Lungu et al. [84] whose system displayed versions of graphs that tried to maintain the locality of the changes. Many other tools attempt to draw temporal graphs of software metrics and changing architecture [101, 105, 120]. We have our own animated graph drawing tool, YARN (Yet Another Reverse-engineering Narrative) [62, 61], which produces videos of the changing call-in and call-out dependencies between modules and shows these dependencies cumulatively over time. For an example of YARN see Figure 6.

Time-lines provide a static, non-animated, visual histogram of counts or amounts over time. Evograph is an MSR related project that maps time to time-lines rather than animations [26]. Time-lines are used in Figure 7 of the Unified process [65] to indicate effort per workflow over time. Note that this diagram of effort is not extracted, it was expected, process recovery could aide in regenerating a diagram like this to be used in a report. Visualization whether it be static graphs, animations or time-lines is useful for exploratory studies of the evolution and process recovery. If a pattern can be spotted visually it should be extractable via automated methods.

2.3.7 Social Aspects

Some studies of software repositories focus mostly on the developers and users, and the communication between developers and users. Much of the social related MSR research analyzes mailing-list repositories [106, 7, 8, 10].

Rigby et al. [106] attempted to characterize the conversational tone of developers on the Apache mailing-list using psychometric textual analysis which contained word lists associated with certain emotional aspects.

Most other socially related MSR research dealt with social network analysis, the network of communications, dependencies and relations between developers, their artifacts and the users. Bird et al. [7, 8, 10, 9] create dynamic and static social network graphs and analyze them for behaviours like when a group of developers enters or a leaves a project, when a user becomes a developer, who becomes a developer, etc. Social network analysis and psychometric analysis are useful for relating stakeholders, such as developers to a time-line, their artifacts and their coworkers artifacts. The study of collaboration gives us hints to how the team works together and maybe their underlying roles and processes.

2.3.8 Concept Location and Topic Analysis

Often when mining various data-sources, one wants to associate high level concepts and ideas with fine-grained entities, *concept analysis*, *concept location* and *topic analysis* help with this task. *Concept analysis*

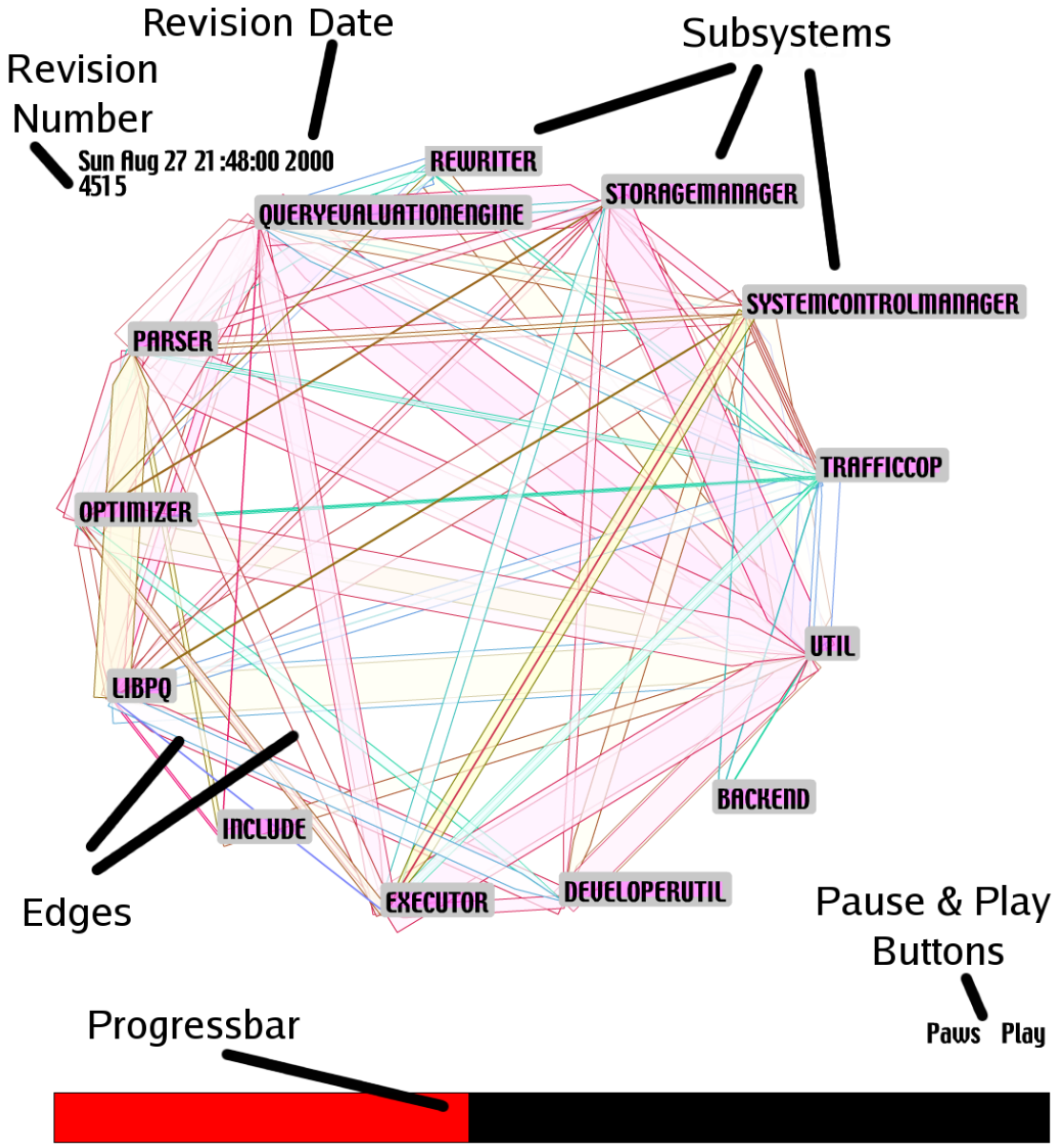


Figure 6: YARN Ball: PostgreSQL module coupling animation, the thickness of the arc and direction indicates how many times one module calls another. Modules are the boxes along the edge of the circle [61].

finds concepts within documents and tries to relate them to source code. *Concept location* takes an already known high level concept such as a bug and tries to find the documents that relate to that concept. For example, when fixing a bug, how and where do the concepts in the bug report map back to the source code? *Topic analysis* attempts to find common topics, word distributions, which reoccur in discussions extracted from documents such as messages, change-logs, or even source code.

Research on formal concept analysis and concept location [86, 87, 102, 103] has often utilized LSI or semantic clustering [72, 73], while topic analysis has used both LDA and LSI [80, 116, 102] For instance, Lukins et al. [116], used LDA for bug localization. They would produce an initial model and then use a bug report as an example of a document they wanted to see. The example document would be broken down into a linear combination of topics and then the most similar documents would be retrieved. Grant et al. [37] and have used an alternative technique, called Independent Component Analysis [22] to separate topic signals from source code. We used LDA to analyze change-log comments [60] in a windowed manner where we apply LDA to windows of changes and relate those windows via similar topics.

Concept location and topic analysis are particularly useful for finding the relationships between work flows like bug fixing or maintenance and the artifacts that mention them. This kind of analysis applied in a MSR setting allows us to generalize about the topics and concepts being tackled within a project.

2.3.9 Summary

Mining software repositories is inherently linked with data analysis and process. Much MSR research attempts to model and explain behaviours and the resulting programs that come from this evolution of artifacts. More importantly MSR is generally the application of different data analysis tools in order to mine artifacts extracted from software repositories. These software repositories range from bug trackers, to source control systems, to mailing-lists. Software process recovery attempts to federate many of these techniques for the purposes of modelling processes from this data.

2.4 Software Process Recovery

Software process recovery, as explained in the introduction, is a sub-field of mining software repositories dedicated to extracting processes, at different levels of granularity, from software repositories. Software process recovery is related to two fields outside of MSR: process mining and process discovery. Process mining attempts to discover business processes from already running formal and informal processes within an organization. Process discovery attempts to apply process mining to software by modifying the development process in order to build up metrics which can be used as evidence of the underlying process. Software process recovery takes a different approach, in order to discover process, one analyzes the existing artifacts left behind by stakeholders such as developers and users.

2.4.1 Process Mining: Business Processes

Business process analysis is a field of process recovery of business processes from existing systems. Researchers such as Van der Aalst [123], attempt to mine workflows and business processes while the process occurs within an organization. Process mining has its own term for fine grained analysis: *Delta analysis*. Much of process mining includes workflow analysis, that consists of applying or extracting finite state machines or petri nets from the observed activities. Process mining attempts to model processes using states extracted from observations.

Some within the MSR community have worked on extracting business processes from software [40, 39, 38]. We find that business processes are too fine-grained for many software processes, we find that they are about the purpose of the software, and not the development of the software itself. Although process mining is quite relevant, process recovery seeks to analyze the development of the software.

2.4.2 Process Discovery

Process discovery is an attempt to take process mining and apply it to software by modifying the existing process to record metrics used to discover underlying processes. In the field of process discovery Cook [16]. has described frameworks for event-based process data analysis [18]. In [19], Cook uses multiple methods for correlation and measurement are used on processes. Various metrics used were string distances between a process model and the actual process data, workflow modelling, and Petri-nets. Cook also discusses grammar inference, neural networks and Markov models as applied to process discovery [17]. The value of process discovery is that much of the groundwork for models of processes for software development have already been implemented. Process recovery is based on process discovery but with a slight twist, instead of changing an already existing process, it seeks to recover process from already existing artifacts.

2.4.3 Process Recovery

By taking process discovery and applying it to software repositories and artifacts already left behind by developers we have process recovery (which was heavily discussed in the introduction). It is a mix of MSR, process mining and process discovery methods all-in-one and applied to information about the past.

Within the MSR community many have tried to extract processes. For instance, Ripoche and Gasser [107] have investigated Markov models for use in OSS bug repair. They created state transition diagrams that represented the probabilistic state transition model of the Bugzilla bug database for the Mozilla project. Their ideas are derived from the process modelling ideas of Cook [16], specifically the idea of using Markov chains to describe the state transitions. This kind of process is closely related to business processes in terms of detail.

Whereas software process recovery at a software development process level has been discussed by Jensen and Scacchi [66]. They describe various ways of mining information from OSS projects to facilitate process modelling. The focus in this line of research was mining web resources to "discover workflows" rather than source control repositories. They use ontologies and directed graphs to describe work flows. Most importantly Jensen et al. uses *a priori* data, that is if they know something, they provide it to their tools and their analysis. In other research, German manually mined process documentation and mailing-lists in order to describe the processes that GNOME project developers used to create and manage the GNOME desktop environment [31].

Thus there has been a range of MSR-related investigation into process recovery. The research has ranged from the automatic to the manual. Different researchers have extracted processes from bug repositories, project websites, and mailing-lists. What is left to be done is a more general integration and general approach to finding iterations, to finding sub-phases, to identifying process within a projects artifacts.

2.4.4 Summary

We have shown three different but related branches of research, starting at the fine-grained level with process mining of business processes, discussed by van der Aalst et al. [123], which are often too fine grained to be useful for software, followed by process discovery, discussed by Cook et al. [16], and then process recovery research [107, 66, 31, 40, 39, 38] from within the mining software repositories community. What distinguishes software process recovery from process discovery and process mining is that process discovery and process mining modify the process of the project it studies or at least watches the process as it executes, while software process recovery relies on the artifacts left behind.

3 Research Proposal

Our overall research plan is to design, implement, and validate a unified methodology for semi-automated process recovery. In part, will extend techniques, tools and results that have already been performed.

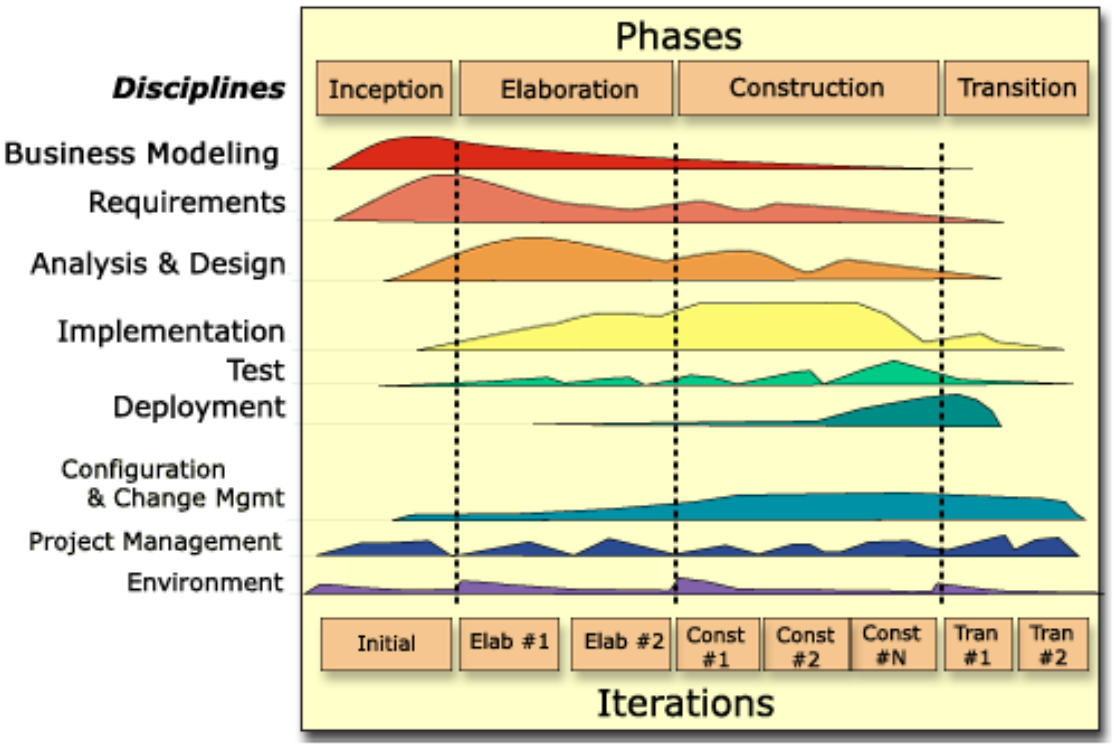


Figure 7: Rational Unified Process (c) 2003 Rational Software Corporation

We plan to extract information about behaviours, repeating behaviours, topics and focus, the underlying life cycle, and processes from development artifacts. These artifacts include mailing list messages, bug tracker events, source control repositories, source code, documentation, test cases, build systems, etc.

One important aspect of our approach will be to give an overview of the emphasis or effort spent on different behaviours. This can be characterized as a "dynamic Rational Unified Process diagram" as shown in 7. This diagram shows the theoretical effort developers put into each discipline or workflow over time, this diagram was meant to communicate the idea rather than impart any real data. Instead of being a theoretical model of a project, this a dynamic version of this diagram can illustrate what the actual and proportional work is being done. Yet it is even more than that, it shows multiple data sources (in this case effort) across time. We are supposed to look at an iteration or phase and then see what kinds of effort correlate with that phase.

The Rational unified process diagram (Figure 7) shows emphasis and effort categorized by workflow across time. This kind of plot would be immediately useful to anyone seeking a high-level overview of where the effort was being spent on a project. This is a kind of reflection of the behaviour and process that goes into a project. We hope that our research in process recovery can help elucidate how this diagram could look for other projects, with little interaction from the end-user.

3.1 What are the processes?

The processes we are looking for are stochastic processes, business processes and software development processes.

Stochastic processes these are models of measurements. Stochastic processes can be represented by functions, distributions, or probability distributions. These are probably the easiest models to extra as often

they simply measurements of the underlying data. Stochastic processes can also include frequency analysis models of data or associations of concepts as well.

Business processes are processes dedicated to fulfilling a business objective. They are processes which are defined as steps are broken down into states and transitions between states. Van der Aalst et al. [123] have represented these models as Markov models, or petri-nets. Generally they are modelled by states and transitions. These models can be extracted from stochastic models via mixture models that represent states. States can then be sequence mined in larger states.

Software development processes are processes which describe how to develop software in a systematic process oriented manner. These processes are more high level that business processes but also describe a different kind of process. Software development processes give indications of effort or focus per workflow. This is similar to the level of detail shown in the UP diagram in Figure 7 or finer. Software development process states can be extracted in a similar manner to business process states, one can model a phase as a general mixture of efforts. By identifying phases one can sequence mine the phases and discover different kinds of iterations. In the end a software development process will look like a state machine of transitions between phases which make up an iteration.

Thus we can see that different kinds of process almost form of a hierarchy in terms of complexity and the scope of the process. The stochastic processes describe the underlying data, the business processes describe immediate states and their transitions, while software development processes describe the phases and iterations of development within a software project.

3.2 How are we going to recover these processes?

We will investigate process in a hierarchical context. We have four levels of the hierarchy, each higher level inheriting the processes, models, and tools of the levels below it. Iteration is the top most level and covers all the previous levels, but is composed of all of the previous levels. Fine-grained level is the bottom-most level and deals with actual measurements and observations, its primary processes are stochastic. The levels in the hierarchy are: fine-grained level, aggregate level, phase level and finally iteration level. Essentially we start at the lowest level, the fine grained level, and work our way up by aggregating models all the way up to the iteration level.

Fine grained level consists of low level events: changes to a file, changes to a file type, changes to source code, communications such as emails, and the immediate relationships between artifacts. Fine grained levels are concerned mostly with measurements, events and observations, thus most processes at this level will be stochastic processes. They can be represented by statistical models, functions, and distributions. The main tools used here are summary statistics, distributions, distribution tests, and immediate coupling metrics.

Aggregate level is above fine grained level and consists of aggregations of the measurements, observations and models within the fine grained view. Analysis at this level would take fine grained views as input and produce higher level aggregates such as models or general effort statistics. These aggregate views can be combined further with other aggregates such that phases or sub-iterations can be identified. The processes that are extractable at this level include stochastic processes but also include business processes and some lightweight software development processes (such as those on how the state of a bug report changes [107]). The models used for these processes include more state-based and Markovian models like those proposed in Cook's process discovery work [16]. Other models include mixture models of observations. States can be created from observations that occur within certain ranges. Other possible process models include sequences. The tools that could be used include machine learning, sequence mining, NLP, time-series analysis and social network analysis. The aggregate level is the workhorse analysis level, providing medium level aggregations in order to aid reasoning, association of artifacts and generation of new facts from old facts. The aggregate level also relies heavily on the co-occurrence of events from different data-sources or data-streams within the

same data-source. This correlation of behaviour is a fundamental part of characterizing process. The next level, phase level, aggregates models and information from this level further.

Phase level attempts to take the results from the aggregate level and start reasoning about the results at a software engineering process level. Given any business processes or stochastic processes found at lower levels, the phase level analysis organizes these models and results from the lower levels into possible phases of software development processes. These phases simply might be mixtures of various behaviours all at once but in different proportions per each phase. An example of phase level analysis includes our work on release pattern discovery where we characterize the behaviour within a project around release time by the proportion of changes related to source code, test code, build code or documentation that occur during that time [58]. Self-similarity [57], item-set partitioning [112], and sequence mining can be used to recover phases and strings of phases, which could be later aggregated into iterations.

Iteration level attempts to mine the software development process of the entire project as well as sub-processes discovered in lower levels of analysis. In general iteration level analysis is all about finding sequences of phases and as well the interaction of processes between different tools and actors. A useful tool for iteration level analysis is item-set partitioning [112] can help with this. Phase transitions can form finite state machine models of the underlying behaviour and processes followed by developers. Iterations can be identified by repeated sub-sequences of sub-iterations below them. These iterations can be partitioned by releases which are previously marked or discovered by partitioning methods like item-set partitioning.

We will investigate process in a hierarchical manner The hierarchy is from ground-up as events and measurements are organized into higher and higher level entities until we reach aggregations of both phases and iterations. These patterns of behaviour will be combined to describe the behaviours exhibited and processes followed by developers while working on software project. Much of this work can be combined into an semi-automated or automated tool.

3.3 Time-line Correlation Methodology

Beyond the analysis methods referenced above in Sections 3.1 and 3.2, we want to create investigate methods to allow for automated or semi-automated correlation analysis to further the state of the art of research in this domain. It would integrate much of work that we have already done [57, 58, 52] with time-line querying utilities in order to mine correlations, mine business processes, phases, iterations and software development processes. We believe what joins all of these data-sets, metrics, and aggregates is that they are all related to each other throughout time. There are probably direct time correlations (events that occur at the same time) and lagged time correlations (one event is ahead of another by some delay).

The scope of the methodology is to allow exploration of data-sets related to software process recovery while allowing both querying and reasoning. Potentially this methodology could be useful to those stakeholders who seek a high level view of the project. But the main focus will be to further process recovery research so that we can build upon what has already been done via this integration work.

We propose to build a tool, a proof of concept, which allows us to:

- automatically generate a workflow effort diagram much like the RUP diagram 7.
- provide us with a time-line form view of many data-sources as to allow visual exploration of potential time correlations.
- allow us to test statistically for time correlations between different time-related data-sources.
- help mine stochastic processes from artifacts.
- help mine business processes from artifacts.
- help mine software develop processes from artifacts.

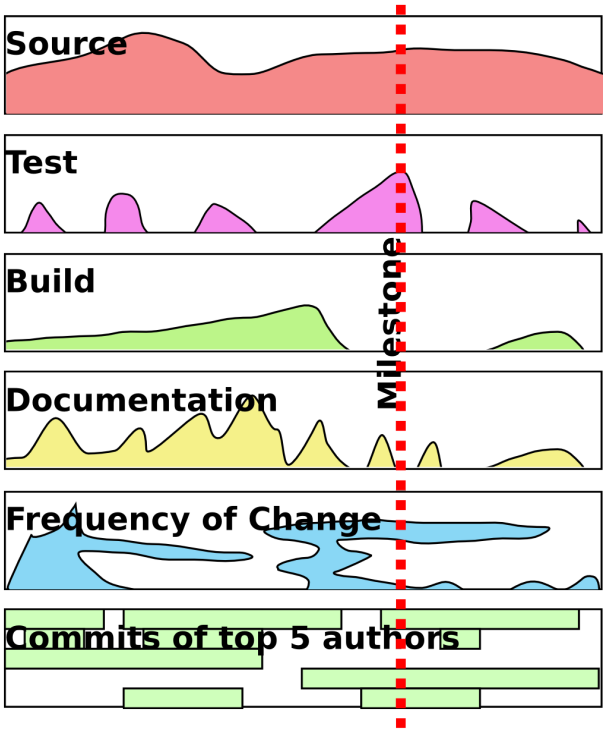
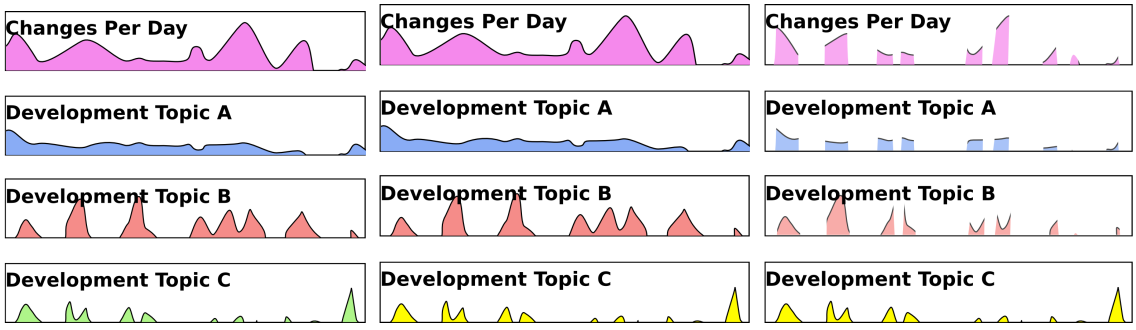


Figure 8: Example time-line view of a software project



(a) Basic view of time-lines

(b) Select a time-series to correlate with

(c) See the values in other data streams that correlate

Figure 9: Example of how time-line querying and time-series correlation could be used.

The workflows that we can mine include changes and effort related to source code, testing, build files and configuration management, and documentation [58]. We also plan to mine communication events like bug tracker events, design documentation, and mailing lists messages. Using topic clustering techniques like LDA [11, 60] and simple NLP [69, 52] techniques we can try to relate communication artifacts and events to other concepts related to software development and the project itself.

Figure 8 serves a static view of a project, as well as a prototypical example of how our propose tool would operate. We would view change events sliced into data streams. Figure 9 provides another prototypical view of how time correlated querying could be visually implemented across different data sources. We could both visualize the results and provide rating of the general correlation.

Throughout the investigation of topics and the correlation of activities to effort we can characterize phases as mixture models of topics and effort. We can then apply sequence mining [68] in order to further abstract sets of phases into a total iteration.

As described in Section 3.1 we aim to extract three kinds of processes: stochastic processes, business processes, and software development processes. We intend to implement this using the analysis techniques discussed in Section 3.2.

Thus we seek to produce a tool will help us:

- Visualize and analyze data about a project with respect to entities and time
- Produce a tool to act as dashboard and time-line of the topics and behaviours of a project
- Produce a methodology and tool that can communicate project activities to stake-holders
- Produce a tool that is able to report on the underlying processes and kinds of processes that are observable.

3.4 Underlying Assumptions

In many of these cases we assume that enough information is left behind that we can reason about the process itself. We are also making the assumption that much of the information recorded is honest and accurate. Unfortunately some MSR research has shown that records are not always reliable [2]. We are also making the assumption that enough process related information can be recorded within the artifacts that developers leave behind.

Another problem we face is that we might be extracting observable processes from the software repository's use, and not necessarily the project's development. What we see simply might be the processes of how one uses a version control system versus the actual development processes. Unfortunately given the scope of our problem this is what we have to work with.

3.5 Validation

Validation of our methods and tools will be split into three main parts: exploratory case studies, validating case study of processes extracted from projects with known processes, and validating case study of an artificially created project which was made with a specific process in mind.

The exploratory case studies will attempt to recover the processes from projects that we have little familiarity with. The intention here is to avoid biasing the results to looking for what we know is there. The resulting processes extracted from this case study could be validated if it was possible to communicate to the developers. This kind of case study shows that we mine valuable facts from unknown projects. The next step would be to validate against projects that are familiar with.

The validating case studies where we extract process from known projects will be used to confirm that our methods and tools can indeed discover processes that we intended to discover. This will investigate MSR "benchmark" projects such as Eclipse, Apache and Mozilla. These case studies would demonstrate that we can extract processes from MSR "benchmark projects" and meet our own expectations, but we still need to test further.

Finally we will generate artificial test-case projects which can be tailored to particular processes or meant to tailored to no process whatsoever. Examples would be to generate revision events which follow a certain distribution, events generated that form expected bug tracking sequences and processes, events generated from state-machines, etc. The non-process project could contain random data or a mixture of many projects in one where some of the data could be legitimate but the rest is garbage. It is necessary to test the garbage case to ensure we are not making up results or simply fishing. These case studies would serve as a test suite for the tool itself.

4 Conclusions

Software process recovery is the recovery of software development processes from artifacts of development that developers leave behind when building and maintaining a software project. These artifacts can be source code, documentation, changes to artifacts, test suites, wikis, mailing-list messages, project websites etc. We have shown that software process recovery relates and contrasts with process mining and process discovery.

Software process recovery is differentiated from previous work by Van der Aalst [123] on process mining because it is about software development processes and not business processes. We have shown that business processes are much more detailed that software development processes and thus are not particularly useful when trying to describe a software development process.

Software process recovery is differentiated from Cook's [16] work on *software process discovery* in that software process recovery occurs after artifacts are made, but process discovery attempts to discover process by injecting process measurement tools into already ongoing development practises.

Software process recovery is an attempt to mine artifacts from the development history of a project in order to characterize the underlying processes and what went on even in the absence of true effort data. Using NLP techniques like topic analysis one can relate artifacts to different aspects of the software development process in use on a project.

Software process recovery is posed within the field of mining software repositories but deals with many kinds of data analysis: statistics, machine learning, and stochastic processes, business processes, software development processes. Software process recovery is already being researched within the mining software repositories community.

We propose a hierarchical analysis framework for software process recovery which ranges from fine-grained measurements and aggregates, to mining phases of an iteration, to characterizing the actual iterations within a project based on past observable behaviour. We also propose to make a methodology which focuses on the analysis of time-windowed correlated events. This methodology attempts to resolve what currently needs to be done in the MSR community, which is an integration of various techniques which can show the inter-related aspects of different events and different requirements.

The value of process recovery comes from its ability to mine facts from data which already exists and is created as part of a normal software development life-cycle. This information can be used to recover the processes, to help document underlying processes and validate the processes. This is especially useful for stakeholders who have no been taking part in the direct development of a project.

5 Appendix

5.1 Timeline

Table 2 gives an indication of past related activities to this line of research and completing my PhD, it also provides a future time-line of when research and parts of the thesis are expected to be finished. The estimated end of degree is for Fall 2010, the Comprehensive II exam is expected to be in early October.

Date	Activity	Description
2005	PhD	Started PhD program
2005	PhD	Took 3/4 Required Courses
2005	Publish	SCQL: A formal model and a query language for source control repositories , MSR 2005
2005	Publish	Measuring Fine-Grained Change in Software: Towards Modification-Aware Change Metrics, METRICS 2005
2006	PhD	Took 4/4 Required Courses
2006	PhD	Met Breadth Requirement, Comp I
2007	Publish	Release Pattern Discovery: A Case Study of Database Systems , ICSM 2007
2007	Publish	YARN: Animating Software Evolution , VISSOFT 2007
2007	Publish	Release Pattern Discovery via Partitioning: Methodology and Case Study , MSR 2007
2008	Publish	Reverse Engineering CAPTCHAs, WCRE 2008
2008	Publish	From Indentation Shapes to Code Structures , SCAM 2008
2008	Publish	Reading Beside the Lines: Indentation as a Proxy for Complexity Metrics, ICPC 2008
2008	Publish	What do large commits tell us?: a taxonomical study of large commits , MSR 2008
2008	PhD	Met PhD Seminar requirement
2009	Publish	What's Hot and What's Not: Windowed Developer Topic Analysis , ICSM 2009
2009	Publish	Mining Recurrent Activities: Fourier Analysis of change events , ICSE 2009
2009	Publish	Reading beside the lines: Using indentation to rank revisions by complexity, Science of Programming
2009	Publish	Automatic Classification of Large Changes into Maintenance Categories, ICPC 2009
2009	PhD	Started Comp II
The Future		
Fall 2009	Conference	Present ICSM 2009 Paper
Fall 2009	PhD	Defend Comp II
Fall 2009	PhD	Initiate Thesis
Fall 2009	Publish	Finish and submit Development Topic naming paper "What's in a name"
Winter 2010	Publish	Finish and submit Multilevel Modelling Work
Spring 2010	Conference	MSR 2010 Challenge Chair
Spring 2010	PhD	Submit drafts of PhD Thesis
Fall 2010	PhD	Finish and Defend PhD Thesis

Table 2: Timeline of Abram Hindle's PhD and publications.

5.2 Related Publications

These are publications that I have contributed to which are directly relevant to this research proposal. There were covered in previous section but are highlighted here as an example of the work that I have already completed in this area.

"SCQL: A formal model and a query language for source control repositories" [51], published at MSR 2005, was a summary of my masters work, that was a source control query language, which is directly relevant to software repositories.

"YARN: Animating Software Evolution" [62], published at VISSOFT 2007, visualized the growing dependencies between modules at an inter-module level. YARN is relevant as it tracks changes at the architectural level over time and thus can indicate product stability.

"Measuring Fine-Grained Change in Software: Towards Modification-Aware Change Metrics" [33], published at METRICS 2005, was work done by Daniel German and myself about metrics which were aware of change, rather than just the difference between before and after metrics. These metrics were meant to operate on changes and revisions. This work related to the three papers I have published on measuring the indentation within revisions to source code: "Reading Beside the Lines: Indentation as a Proxy for Complexity Metrics" [56], in ICPC 2008; "From Indentation Shapes to Code Structures" [55], in SCAM 2008 ; and "Reading beside the lines: Using indentation to rank revisions by complexity" [59] in the Science of Programming. All of these metrics are directly related to this work because they operate at a fine grained level on software repository data.

"Release Pattern Discovery: A Case Study of Database Systems" [54], published at ICSM 2007, tracked developer behaviour around release time. It followed "Release Pattern Discovery via Partitioning: Methodology and Case Study"[58], which was published at MSR 2007. This work is quite central to this proposal as it tries to characterize developer behaviour.

We studied classifying commits by their maintenance types in these papers: "What do large commits tell us?: a taxonomical study of large commits" [53], in MSR 2008; and "Automatic Classification of Large Changes into Maintenance Categories" [52], in ICPC 2009. These studies revolved around understanding the distributions of the underlying purposes behind the revisions and applying machine learning to classify the purpose of new revisions.

"Mining Recurrent Activities: Fourier Analysis of change events" [57], published at ICSE 2009 in the NEIR track, was about the use of the Fourier transform to analyze change data from repositories. This kind of analysis enables repeating events to be discovered. Determining recurrent activity is valuable as it implies some kind of underlying process.

"What's Hot and What's Not: Windowed Developer Topic Analysis" [60], published at ICSM 2009, attempts to discover repeating topics that occur during the development of a project. This kind of information allows us to discover the issues being addressed during a certain period. Topic patterns might allow us to characterize the repeating responsibilities that programmers face on a project.

Thus as you can see I have published much relevant work in the area and plan to utilize this work in order to come up with a coherent methodology for semi-automatic process recovery.

References

[1] G. Antoniol, V. F. Rollo, and G. Venturi. Linear predictive coding and cepstrum coefficients for mining time variant information from software repositories. In *MSR '05: Proceedings of the 2005 international workshop on Mining software repositories*, pages 1–5, New York, NY, USA, 2005. ACM.

[2] J. Aranda and G. Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 298–308, May 2009.

[3] T. Ball, J.-M. K. Adam, A. P. Harvey, and P. Siy. If your version control system could talk. In *ICSE Workshop on Process Modeling and Empirical Studies of Software Engineering*, 1997.

[4] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 1st edition, October 1999.

[5] D. Bellin, M. Tyagi, and M. Tyler. Object-oriented metrics: an overview. In *CASCON '94: Proceedings of the 1994 conference of the Centre for Advanced Studies on Collaborative research*, page 4. IBM Press, 1994.

[6] D. Beyer and A. E. Hassan. Animated Visualization of Software History using Evolution Storyboards. In *WCRE 2006*, pages 199–210, 2006.

[7] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining Email Social Networks. In *Proceedings of the Third International Workshop on Mining software repositories*, pages 137–143. ACM, 2006.

[8] C. Bird, A. Gourley, P. Devanbu, A. Swaminathan, and G. Hsu. Open Borders? Immigration in Open Source Projects. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 6. IEEE Computer Society, 2007.

[9] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy. Putting it All Together: Using Socio-Technical Networks to Predict Failures. In *Proceedings of the 17th International Symposium on Software Reliability Engineering*. IEEE Computer Society, 2009.

[10] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu. Latent Social Structure in Open Source Projects. In *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 24–35. ACM, 2008.

[11] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003.

[12] B. Boehm. A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes*, 11(4):14–24, 1986.

[13] J. Brichau, C. De Roover, and K. Mens. Open unification for program query languages. In *Chilean Society of Computer Science, 2007. SCCC '07. XXVI International Conference of the*, pages 92–101, Nov. 2007.

[14] A. Capiluppi, P. Lago, and M. Morisio. Characteristics of open source projects. *csmr*, 00:317, 2003.

[15] D. Coleman, D. Ash, B. Lowther, and P. W. Oman. Using metrics to evaluate software system maintainability. *Computer*, 27(8):44–49, 1994.

[16] J. E. Cook. *Process Discovery and Validation through Event-Data Analysis*. PhD thesis, Computer Science Dept., University of Colorado, 1996.

[17] J. E. Cook and A. L. Wolf. Automating process discovery through event-data analysis. In *ICSE '95: Proceedings of the 17th international conference on Software engineering*, pages 73–82, New York, NY, USA, 1995. ACM Press.

[18] J. E. Cook and A. L. Wolf. Balboa: A framework for event-based process data analysis. In *Proc. of the 5th International Conference on the Software Process*, pages 99–110, June 1998.

[19] J. E. Cook and A. L. Wolf. Software process validation: quantitatively measuring the correspondence of a process to a model. *ACM Trans. Softw. Eng. Methodol.*, 8(2):147–176, 1999.

[20] D. Cubranic and G. C. Murphy. Hipikat: Recommending pertinent software development artifacts. In *Proceedings of the 2003 International Conference on Software Engineering*, pages 408–418, Portland, 2003. Association for Computing Machinery.

[21] M. D'Ambros and M. Lanza. Reverse Engineering with Logical Coupling. In *WCRE*, pages 189–198, 2006.

[22] K. Delac, M. Grgic, and S. Grgic. Independent comparative study of PCA, ICA, and LDA on the FERET data set. *International Journal of Imaging Systems and Technology*, 15(5):252–260, 2006.

[23] D. Draheim and L. Pekacki. Process-centric analytical processing of version control data. In *Sixth International Workshop on Principles of Software Evolution (IWPSE'03)*, pages 131–136. IEEE, 2003.

[24] S. Ducasse, M. Lanza, A. Marcus, J. I. Maletic, and M.-A. D. Storey, editors. *Proceedings of the 3rd International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT 2005, September 25, 2005, Budapest, Hungary*. IEEE Computer Society, 2005.

[25] N. E. Fenton. *Software Metrics: A Rigorous Approach*. Chapman & Hall, Ltd., 1991.

[26] M. Fischer and H. Gall. EvoGraph: A Lightweight Approach to Evolutionary and Structural Analysis of Large Software Systems. In *WCRE*, pages 179–188, 2006.

[27] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *Proceedings of the International Conference on Software Maintenance (ICSM 2003)*, pages 23–32, 2003.

[28] H. Gall, M. Jazayeri, , and J. Krajewski. CVS Release History Data for Detecting Logical Couplings. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, September 1998.

[29] H. Gall, M. Jazayeri, and J. Krajewski. CVS Release History Data for Detecting Logical Couplings. In *Proc. of the International Workshop on Principles of Software Evolution (IWPSE)*, pages 12–23, 2003.

[30] H. Gall, M. Jazayeri, and C. Riva. Visualizing software release histories: The use of color and third dimension. In *ICSM '99: Proceedings of the IEEE International Conference on Software Maintenance*, page 99, Washington, DC, USA, 1999. IEEE Computer Society.

[31] D. M. German. Decentralized open source global software development, the GNOME experience. *Journal of Software Process: Improvement and Practice*, 8(4):201–215, 2004.

[32] D. M. German. Mining CVS repositories, the softChange experience. In *1st International Workshop on Mining Software Repositories*, pages 17–21, May 2004.

[33] D. M. German and A. Hindle. Measuring fine-grained change in software: towards modification-aware change metrics. In *Proceedings of 11th International Software Metrics Symposium (Metrics 2005)*, 2005.

[34] D. M. German, A. Hindle, and N. Jordan. Visualizing the evolution of software using softchange. In *Proceedings SEKE 2004 The 16th Internation Conference on Software Engineering and Knowledge Engineering*, pages 336–341, 3420 Main St. Skokie IL 60076, USA, June 2004. Knowledge Systems Institute.

[35] T. Gîrba, S. Ducasse, and M. Lanza. Yesterday's weather: Guiding early reverse engineering efforts by summarizing the evolution of changes. In *ICSM*, pages 40–49, 2004.

[36] M. W. Godfrey and Q. Tu. Evolution in Open Source Software: A Case Study. In *Proceedings of International Conference on Software Maintenance*, pages 131–142, 2000.

[37] S. Grant, J. R. Cordy, and D. Skillicorn. Automated concept location using independent component analysis. In *15th Working Conference on Reverse Engineering*, 2008.

[38] J. Guo, K. C. Foo, L. Barbour, and Y. Zou. A business process explorer: recovering and visualizing e-commerce business processes. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 871–874, New York, NY, USA, 2008. ACM.

[39] J. Guo and Y. Zou. A business process explorer: Recovering business processes from business applications. In *Reverse Engineering, 2008. WCRE '08. 15th Working Conference on*, pages 333–334, Oct. 2008.

[40] J. Guo and Y. Zou. Detecting clones in business applications. In *Reverse Engineering, 2008. WCRE '08. 15th Working Conference on*, pages 91–100, Oct. 2008.

[41] M. H. Halstead. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., New York, NY, USA, 1977.

[42] R. Harrison, S. Counsell, and R. Nithi. An evaluation of the mood set of object-oriented software metrics. *Software Engineering, IEEE Transactions on*, 24(6):491–496, Jun 1998.

[43] D. Hartmann and R. Dymond. Appropriate agile measurement: Using metrics and diagnostics to deliver business value. In *AGILE '06: Proceedings of the conference on AGILE 2006*, pages 126–134, Washington, DC, USA, 2006. IEEE Computer Society.

[44] A. Hassan. Mining software repositories to guide software development. `http://plg.uwaterloo.ca/~aeehassa/home/pubs.html`, 2005. Accessed July.

[45] A. E. Hassan and R. C. Holt. C-REX: An Evolutionary Code Extractor for C - (PDF). Technical report, University of Waterloo. http://plg.uwaterloo.ca/ aeehassa/home/pubs/crex.pdf.

[46] A. E. Hassan and R. C. Holt. Predicting change propagation in software systems. In *Proceedings of ICSM 2004: International Conference on Software Maintenance*, pages 284–293, September 2004.

[47] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles. Forecasting the number of changes in eclipse using time series analysis. In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 32, Washington, DC, USA, 2007. IEEE Computer Society.

[48] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles. Towards a theoretical model for software growth. In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 21, Washington, DC, USA, 2007. IEEE Computer Society.

[49] J. Highsmith and M. Fowler. The agile manifesto. *Software Development Magazine*, 9(8):29–30, 2001.

[50] E. Hill. Developing natural language-based program analyses and tools to expedite software maintenance. In *ICSE Companion '08: Companion of the 30th international conference on Software engineering*, pages 1015–1018, New York, NY, USA, 2008. ACM.

[51] A. Hindle and D. M. German. Scql: a formal model and a query language for source control repositories. In *MSR '05: Proceedings of the 2005 international workshop on Mining software repositories*, pages 1–5, New York, NY, USA, 2005. ACM.

[52] A. Hindle, D. M. German, M. W. Godfrey, and R. C. Holt. Automatic classification of large changes into maintenance categories. In *International Conference on Program Comprehension*, Vancouver, May 2009. in press.

[53] A. Hindle, D. M. German, and R. Holt. What do large commits tell us?: a taxonomical study of large commits. In *MSR '08: Proceedings of the 2008 international working conference on Mining software repositories*, pages 99–108, New York, NY, USA, 2008. ACM.

[54] A. Hindle, M. Godfrey, and R. Holt. Release pattern discovery: A case study of database systems. In *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, pages 285–294, Oct. 2007.

[55] A. Hindle, M. Godfrey, and R. Holt. From indentation shapes to code structures. In *8th IEEE Intl. Working Conference on Source Code Analysis and Manipulation (SCAM 2008)*, 09 2008.

[56] A. Hindle, M. Godfrey, and R. Holt. Reading beside the lines: Indentation as a proxy for complexity metrics. In *Proceedings of ICPC 2008*, June 2008.

[57] A. Hindle, M. Godfrey, and R. Holt. Mining recurrent activities: Fourier analysis of change events. In *Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, pages 295–298, May 2009.

[58] A. Hindle, M. W. Godfrey, and R. C. Holt. Release pattern discovery via partitioning: Methodology and case study. In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 19, Washington, DC, USA, 2007. IEEE Computer Society.

[59] A. Hindle, M. W. Godfrey, and R. C. Holt. Reading beside the lines: Using indentation to rank revisions by complexity. *Science of Computer Programming*, 74(7):414 – 429, 2009. Special Issue on Program Comprehension (ICPC 2008).

[60] A. Hindle, M. W. Godfrey, and R. C. Holt. What's hot and what's not: Windowed developer topic analysis. In *International Conference on Software Maintenance*, 2009.

[61] A. Hindle, Z. Jiang, W. Koleilat, M. W. Godfrey, and R. C. Holt. Yarn ball example, 2007. http://swag.uwaterloo.ca/~ahindle/yarn/postgres.html.

[62] A. Hindle, Z. M. Jiang, W. Koleilat, M. Godfrey, and R. Holt. Yarn: Animating software evolution. In *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on*, pages 129–136, June 2007.

[63] T. Hofmann. Probabilistic latent semantic indexing. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57, New York, NY, USA, 1999. ACM.

[64] E. F. Ian H.Witten. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. The Morgan Kaufmann Series in Data Management Systems, Jim Gray, Series Editor, October 1999.

[65] I. Jacobson, G. Booch, and J. Rumbaugh. *The unified software development process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[66] C. Jensen and W. Scacchi. Simulating an automated approach to discovery and modeling of open source software development processes. In *Proceedings of the Third Workshop on Open Source Software Engineering ICSE03-OSSE03*, May 2003.

[67] H. Kagdi, M. L. Collard, and J. I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *J. Softw. Maint. Evol.*, 19(2):77–131, 2007.

[68] H. Kagdi, S. Yusuf, and J. I. Maletic. Mining sequences of changed-files from version histories. In *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*, pages 47–53, New York, NY, USA, 2006. ACM.

[69] P. Kantor. Foundations of statistical natural language processing. *Inf. Retr.*, 4(1):80–81, 2001.

[70] C. F. Kemerer and S. Slaughter. An Empirical Approach to Studying Software Evolution. *IEEE Transactions on Softwware Engineering*, 25(4):493–509, 1999.

[71] M. Kim, D. Notkin, and D. Grossman. Automatic inference of structural changes for matching across program versions. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 333–343, Washington, DC, USA, 2007. IEEE Computer Society.

[72] A. Ko, B. Myers, and D. H. Chau. A linguistic analysis of how people describe software problems. *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on*, pages 127–134, Sept. 2006.

[73] A. Kuhn, S. Ducasse, and T. Girba. Enriching reverse engineering with semantic clustering. *Reverse Engineering, 12th Working Conference on*, pages 10 pp.–, Nov. 2005.

[74] H.-J. Kung. Quantitative method to determine software maintenance life cycle. In *ICSM*, pages 232–241. IEEE Computer Society, 2004.

[75] M. Lehman, D. Perry, J. Ramil, W. Turski, and P. Wernick. Metrics and laws of software evolution-the nineties view. In *Proceedings of Metrics Symposium 1997*, Nov 1997.

[76] M. M. Lehman. Programs, life cycles and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, 1980.

[77] M. M. Lehman. Laws of software evolution revisited. In *EWSPT '96: Proceedings of the 5th European Workshop on Software Process Technology*, pages 108–124, London, UK, 1996. Springer-Verlag.

[78] M. M. Lehman, D. E. Perry, J. F. Ramil, W. M. Turski, and P. D. Wernick. Metrics and laws of software evolution - the nineties view. In *Metrics '97, IEEE*, pages 20–32, 1997.

[79] S. Limited. Semmle company website. `http://semmle.com`.

[80] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi. Mining concepts from code with probabilistic topic models. In *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 461–464, New York, NY, USA, 2007. ACM.

[81] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi. Mining eclipse developer contributions via author-topic models. *Mining Software Repositories, International Workshop on*, 0:30, 2007.

[82] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi. Mining internet-scale software repositories. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA, 2008.

[83] Y. Liu and E. Stroulia. Reverse Engineering the Process of Small Novice Software Teams. In *Proc. 10th Working Conference on Reverse Engineering*, pages 102–112. IEEE Press, November 2003.

[84] M. Lungu and M. Lanza. Exploring Inter-Module Relationships in Evolving Software Systems. *CSMR 2007*, 0:91–102, 2007.

[85] A. Marcus, L. Feng, and J. I. Maletic. 3D Representations for Software Visualization. In *SoftVis '03: Proceedings of the 2003 ACM symposium on Software visualization*, pages 27–ff, New York, NY, USA, 2003. ACM Press.

[86] A. Marcus, V. Rajlich, J. Buchta, M. Petrenko, and A. Sergeyev. Static techniques for concept location in object-oriented code. *Program Comprehension, 2005. IWPC 2005. Proceedings. 13th International Workshop on*, pages 33–42, May 2005.

[87] A. Marcus, A. Sergeyev, V. Rajlich, and J. Maletic. An information retrieval approach to concept location in source code. *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*, pages 214–223, Nov. 2004.

[88] T. J. McCabe and C. W. Butler. Design complexity measurement and testing. *Commun. ACM*, 32(12):1415–1425, 1989.

[89] R. Meli. Measuring change requests to support effective project management practices. In *ESCOM Conference*, 2001.

[90] T. Mens and S. Demeyer. Evolution metrics. In *IWPSE '01: Proceedings of the 4th International Workshop on Principles of Software Evolution*, New York, NY, USA, 2001. ACM Press.

[91] T. Mens and S. Demeyer. Future trends in software evolution metrics. In *IWPSE '01: Proceedings of the 4th International Workshop on Principles of Software Evolution*, pages 83–86, New York, NY, USA, 2001. ACM Press.

[92] C. Mesnage and M. Lanza. White Coats: Web-Visualization of Evolving Software in 3D. In Ducasse et al. [24], pages 40–45.

[93] A. Mockus. Amassing and indexing a large sample of version control systems: Towards the census of public source code history. In *Mining Software Repositories, 2009. MSR '09. 6th IEEE International Working Conference on*, pages 11–20, May 2009.

[94] A. Mockus, R. T. Fielding, and J. Herbsleb. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):1–38, July 2002.

[95] H. A. Muller and K. Klashinsky. Rigi - A System for Programming-in-the-large. Technical report, In IEEE 10th International Conference on Software Engineering(ICSE-1998), April 1998.

[96] NIST. *NIST/SEMATECH e-Handbook of Statistical Methods*, 2008. http://www.itl.nist.gov/div898/handbook/.

[97] M. Ogawa, K.-L. Ma, C. Bird, P. T. Devanbu, and A. Gourley. Visualizing Social Interaction in Open Source Software Projects. In *Sixth International Asia-Pacific Symposium on Visualization*, pages 25–32, 2007.

[98] P. W. Oman and J. Hagemeister. Construction and testing of polynomials predicting software maintainability. *J. Syst. Softw.*, 24(3):251–266, 1994.

[99] M. C. Paulk, B. Curtis, E. Averill, J. Bamberger, T. Kasse, M. Konrad, J. Perdue, C. Weber, and J. Withey. *The capability maturity model: guidelines for improving the software process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[100] M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber. Capability maturity model for software. Technical Report CMU/SEI-91-TR-24 ADA240603, 1991.

[101] M. Pinzger, H. Gall, M. Fischer, and M. Lanza. Visualizing Multiple Evolution Metrics. In *Proceedings of the ACM Symposium on Software Visualization*, pages 67–75, St. Louis, Missouri, 2005. ACM Press.

[102] D. Poshyvanyk and A. Marcus. Combining formal concept analysis with information retrieval for concept location in source code. *International Conference on Program Comprehension*, 0:37–48, 2007.

[103] D. Poshyvanyk, A. Marcus, V. Rajlich, Y.-G. Guéhéneuc, and G. Antoniol. Combining probabilistic ranking and latent semantic indexing for feature identification. *International Conference on Program Comprehension*, 0:137–148, 2006.

[104] S. Purao and V. Vaishnavi. Product metrics for object oriented systems. *ACM Computing Surveys*, 35(2), 2003.

[105] J. Ratzinger, M. Fischer, and H. Gall. EvoLens: Lens-View Visualizations of Evolution Data. In *Proceedings of the International Workshop on Principles of Software Evolution*, pages 103–112, Lisbon, Portugal, September 2005. IEEE Computer Society Press.

[106] P. C. Rigby and A. E. Hassan. What can oss mailing lists tell us? a preliminary psychometric text analysis of the apache developer mailing list. *Mining Software Repositories, International Workshop on*, 0:23, 2007.

[107] G. Ripoche and L. Gasser. Scalable automatic extraction of process models for understanding f/oss bug repair. In *Proceedings of the International Conference on Software and Systems Engineering and their Applications (ICSSEA'03)*, December 2003.

[108] J. Rosenberg. Some misconceptions about lines of code. *metrics*, 00:137, 1997.

[109] W. W. Royce. Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th International Conference on Software Engineering*, pages 328–339. IEEE Computer Society Press, 1987.

[110] K. Schwaber and M. Beedle. *Agile Software Development with Scrum*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[111] J. D. Singer and J. B. Willett. *Applied Longitudinal Data Analysis: Modeling Change and Event Occurence*. Oxford University Press, 2003.

[112] H. Siy, P. Chundi, D. Rosenkrantz, and M. Subramaniam. Discovering dynamic developer relationships from software version histories by time series segmentation. In *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, pages 415–424, Oct. 2007.

[113] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? In *MSR '05: Proceedings of the 2005 international workshop on Mining software repositories*, pages 1–5, New York, NY, USA, 2005. ACM.

[114] I. I. Software and R. Singh. International standard. In *Software Lifecycle Process Standards, in CrossTalk*, pages 6–8, 1989.

[115] G. Sridhara, E. Hill, L. Pollock, and K. Vijay-Shanker. Identifying word relations in software: A comparative study of semantic similarity tools. In *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on*, pages 123–132, June 2008.

[116] L. H. E. Stacy K. Lukins, Nicholas A. Kraft. Source code retrieval for bug localization using latent dirichlet allocation. In *15th Working Conference on Reverse Engineering*, 2008.

[117] D. Stelzer, W. Mellis, and G. Herzwurm. A critical look at iso 9000 for software quality management. *Software Quality Control*, 6(2):65–79, 1997.

[118] M.-A. D. Storey, C. Best, and J. Michaud. SHriMP Views: An Interactive Environment for Exploring Java Programs. In *9th International Workshop on Program Comprehension (IWPC 2001), 12-13 May 2001, Toronto, Canada*, pages 111–112. IEEE Computer Society, 2001.

[119] N. Synytskyy, R. C. Holt, and I. Davis. Browsing software architectures with lsedit. In *IWPC '05: Proceedings of the 13th International Workshop on Program Comprehension*, pages 176–178, Washington, DC, USA, 2005. IEEE Computer Society.

[120] A. Telea and L. Voinea. Interactive Visual Mechanisms for Exploring Source Code Evolution. In Ducasse et al. [24], pages 52–57.

[121] W. M. Turski. Reference model for smooth growth of software systems. *IEEE Transactions on Software Engineering*, 22(8):599–600, 1996.

[122] W. M. P. van der Aalst and K. Bisgaard Lassen. Translating unstructured workflow processes to readable bpel: Theory and implementation. *Inf. Softw. Technol.*, 50(3):131–159, 2008.

[123] W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: A survey of issues and approaches. *Data & Knowledge Engineering*, 47(2):237 – 267, 2003.

[124] S. Wasserman, K. Faust, and D. Iacobucci. *Social Network Analysis: Methods and Applications (Structural Analysis in the Social Sciences)*. Cambridge University Press, November 1994.

[125] T. Wolf, A. Schr, D. Damian, L. D. Panjer, and T. H. Nguyen. Mining task-based social networks to explore collaboration in software teams. *IEEE Software*, 26(1):58–66, 2009.

[126] J. Wu and R. C. Holt. Linker-Based Program Extraction and Its Uses in Studying Software Evolution. Technical report, In Proceedings of the International Workshop on Unanticipated Software Evolution (FUSE 2004), March 2004.

[127] A. Zaidman, B. V. Rompaey, S. Demeyer, and A. v. Deursen. Mining software repositories to study co-evolution of production & test code. In *ICST '08: Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation*, pages 220–229, Washington, DC, USA, 2008. IEEE Computer Society.

[128] T. Zimmermann and P. Weisgerber. Preprocessing CVS data for fine-grained analysis. In *1st International Workshop on Mining Software Repositories*, May 2004.