Mining Sequences of Changed-files from Version Histories

Huzefa Kagdi, Shehnaaz Yusuf, Jonathan I. Maletic
Department of Computer Science
Kent State University
Kent Ohio 44242

{hkagdi, sdawoodi, jmaletic}@cs.kent.edu

ABSTRACT
Modern source-control systems, such as *Subversion*, preserve change-sets of files as atomic commits. However, the specific ordering information in which files were changed is typically not found in these source-code repositories. In this paper, a set of heuristics for grouping change-sets (i.e., log-entries) found in source-code repositories is presented. Given such groups of change-sets, sequences of files that frequently change together are uncovered. This approach not only gives the (unordered) sets of files but supplements them with (partial temporal) ordering information. The technique is demonstrated on a subset of *KDE* source-code repository. The results show that the approach is able to find sequences of changed-files.

Categories and Subject Descriptors
D.2.7. [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *documentation, enhancement*, *extensibility, version control*

General Terms
Management, Experimentation

Keywords
Mining Software Repositories, Heuristics, Change Sequences

1. INTRODUCTION
Source-code repositories store metadata such as user-ids, timestamps, and commit comments. This metadata explains the *why*, *who*, and *when* dimensions of a source-code change. Researchers have utilized this type of information for a variety of purposes in the context of supporting and understanding software evolution [5, 6, 9, 12-15] . This includes discovering entities (e.g., files) that frequently change together for the purpose of supporting software-change prediction [3, 8, 10, 11, 16, 17, 21]. Software-change prediction approaches based on itemset mining produce unordered collections of the changed entities. For example, a set of files {f1, f2} that are frequently changed or rules such as changes in a set {f1, f2} leads to changes in a set {f3, f4}.

MSR'06, May 22–23, 2006, Shanghai, China.

However, software changes are inherently (partially) ordered along the time dimension[1]. Itemset mining approaches ignore the ordering information in the mining phase. However, the ordering must be considered at a later stage in software-change prediction. For example if a set of changed-files {f1, f2} is equivalent to a set {interface, implementation}, the changes are not necessarily symmetric. The mined set {f1, f2} may be an artifact of only the interface changes, {f1} leading to implementation changes, {f2} and not vice-versa. Therefore, ignoring the ordering information could lead to a false prediction of {f1} due to a change in {f2}.

Here, we explore the ordering of changed-files by utilizing the information found in the versions log of source-code repositories. We present an approach that processes the log-entries to deduce the partial ordering information among changed-files. For example, our approach discovers sequences of changed-files such as {f1}→{f2} and {f4}→{f5}. The sequence {f1}→ {f2} indicates that changes in {f1} *happens before* {f2}. We term this problem as mining sequences of changed-files. We define six heuristics for grouping the log-entries (i.e., change-sets) of a source-code repository. Given such a group of log-entries we uncover sequences of files that frequently change together. This approach gives not only the (unordered) sets of files but supplements them with (partial) ordering information. Therefore, this approach of changed-files sequence-mining subsumes the approach of changed-files itemset mining.

The rest of the paper is organized as follows. In section 2, we discuss the available change-set records from source-code repositories. In section 3, we present heuristics for grouping. In section 3, we discuss frequent sequence mining. In section 4, we describe the developed toolset. In section 5, we apply our approach on *KDE* version history. In section 6, we briefly discuss related work. Finally, we state our conclusions and future directions in section 7.

2. CHANGE-SETS RECORDS
There is an inherent temporal ordering between various change-sets. It is not uncommon to have a change-set either planned (e.g., a standard refactoring or a fix for a documented bug) or unplanned activity (e.g., a violation of hidden dependencies) leading to further change-sets. First, we examine how these change-sets are recorded in repositories maintained by modern source control systems.

Among several other improvements over CVS and alike, modern source-control systems, such as *Subversion*, preserve the grouping

[1] In the rest of the discussion, ordering and temporal ordering are used interchangeably unless specified.

of several changes in multiple files to a single change-set as performed by a committer (i.e., an atomic commit). Version-number assignment and metadata are associated at the change-set level and recorded as a *logentry*. As shown in Figure 1, a change-set is stored as a single logentry. *Subversion*'s log-entries include the (structured) dimensions *committer*, *date*, and *paths* (i.e., files) involved in a change-set. As shown in Figure 1, each logentry is uniquely identified by a *revision* number. There is no temporal ordering between paths *khtml_part.cpp* and *loader.h*. Clearly, the logentry alone is insufficient to give the temporal ordering of the files involved in a change-set. However, there is a temporal order between change-sets. Change-sets with greater revision numbers occur after those with lesser revision numbers. Therefore, we can utilize the ordering of change-sets to determine ordering of files.

A straightforward approach is to exhaustively list all the sequences of the changed-files. For example, if a change-set {f1, f2} occurs before {f3. f4}, the possible changed-file sequences are {f1}→{f3}, {f1, f2}→{f3}, and so forth. However, this leads to two major issues: 1) sequences that may not be useful for software evolution tasks such as change predication (i.e., false positives) and 2) examination of combinatorial explosion of changed-file sequences. Notice that the atomic commits are serialized. The temporal order in which log-entries appear in the log files is at discretion of a version-control system. As a result successive log-entries may be unrelated in the context of changes performed in the files. Therefore, it may result in meaningless changed-file sequences.

In an effort to avoid reporting of meaningless changed-file sequences, we define heuristics for grouping "related" change-sets. Furthermore, given such related change-sets, we employ sequence mining to effectively deal with the combinatorial explosion of search space.

3. CHANGE-SET GROUPING HEURISTICS

The heuristics are driven by grouping of log-entries based on the dimensions committer, date, and the paths as discussed below.

Time Interval - Change-sets committed in the same time-interval are related and change-sets committed in different time-intervals are unrelated. This helps define ordering on the change-sets in the same time-interval. Therefore, all the change-sets (i.e., log-entries) committed in a given time duration are placed in a single group. The sequences of files found using this heuristic implies that if a file is modified in a sequence on a day, the following (preceding) files are modified on the same day.

Committer – The change-sets modified by a committer are related and the change-sets modified by different committers are unrelated. This defines an order on the change-sets by a committer. Therefore, all the change-sets (i.e., log-entries) committed by a given committer are placed in a single group. The sequences of files found using this heuristic implies that if a file is modified in a sequence by a committer, the following (preceding) files are modified by the same committer.

File – Change-sets involving a particular file are related. This defines ordering on the change-sets by a particular file. Therefore, all the change-sets (i.e., log-entries) committed in which a given file is involved are placed in a single group. The sequences of files found using this heuristic establishes a temporal position of a file in the sequences of changes with other files.

Joins of the above heuristics lead to further groupings of change-sets. For example, the join of heuristics Committer and File implies committers typically do not work on a same change-set. However, committers may work on the same file in different time intervals. This defines an order on the changed-files by a committer in a time interval. The sequences of files found using this heuristic implies that if a file is modified on a day by a committer, the following (preceding) files are modified on the same day by the same committer.

```
<?xml version="1.0" encoding="utf-8"?>
<log>
  <logentry  revision="438663">
    <author>kling</author>
    <date>2005-07-25T17:46:20.434104Z</date>
    <paths>
      <path action="M">khtml_part.cpp</path>
      <path action="M">loader.h</path>
    </paths>
    <msg>
       Do pixmap notifications when
       running ad filters.
    </msg>
  </logentry>
</log>
```

Figure 1. A Snippet of kdelibs Subversion Log

These heuristics are the first step towards mining sequences of changed-file sets. Heuristics helps us to define a logical grouping of change-sets but does not directly give the order in which files were changed within a given change-set. In the next section, we describe our approach of mining frequent sequences of changed-files from grouped change-sets.

4. MINING CHANGED-FILE SEQUENCES

The problem of mining sequences of changed-files is an instance of mining frequent sequences of items. We first give definitions of a sequence and the problem of frequent sequence mining. Then, we show the reduction of our problem to the problem of mining frequent sequences.

A frequent-sequence is made up of (ordered) elements. Each element is made up of (unordered) items. The ordering of elements imposes a partial order on the items. For example, the frequent sequence {f1, f2}→{f3, f4}→{f5} is made up of 3 elements and 5 items. It indicates that the element {f1, f2} happens before the element {f3, f4} and the element {f3, f4} happens before the element {f5}. However, the happens before relation between items f3 and f4 is unknown in the element {f3, f4}. Therefore, a frequent-sequence establishes both the ordered and unordered relationship between items.

The problem of finding frequent sets of sequences is formally defined as given a set of items, $\alpha = \{i_1, i_2, \ldots. i_m\}$, and a set of transactions, $\tau = \{T_1, T_2, \ldots.., T_n\}$, find all the sets of sequences, $S = \{S_1, S_2, \ldots S_o\}$, that co-occur in at least a given number (or percentage) of transactions i.e., it satisfies a given minimum support, σ_{min}. Each Transaction contains an ordered list of events and is identified by an unique id, $T_i = (tid, \varepsilon)$ where $\varepsilon = [E_1, E_2, \ldots., E_p] \mid \forall_{i,j}\ E_i \rightarrow E_j$. and $\rightarrow$ is a given ordering relation on events. Each event contains a set of items and is identified by an unique id, $E_i = (eid, \subseteq \alpha)$. Each sequence is defined as an ordered list of elements (i.e., itemsets), $S_I = [I_1 \rightarrow I_2 \rightarrow \ldots..\rightarrow I_p] \mid \forall I_i \subseteq \alpha$,

and each member of an element, $i_j \in I_i$ is defined as an item of a sequence. A mined sequence S_i is called a *frequent sequence.* A sequence consisting of k items is referred to as a *k-sequence*. The number (or percentage) of transactions in which a sequence occur is known as its *support* (*frequency*).

The problem of frequent-sequence mining was introduced by Agrawal [1]. A number of algorithms for frequent-sequence mining are proposed. Their discussion is out of the scope in this paper. The reduction of our problem of mining sequences of changed-files to that of frequent-sequence mining is straightforward. Here, we have a set of transactions, τ, mapped to the grouping of change-sets formed by the application of a heuristic on the log-entries, events $E_i \rightarrow \ldots \rightarrow E_j$ in each transaction, T_i, maps to the change-sets ordered by revision numbers. Following this reduction, the solution to the frequent-sequence problem, S, gives the solution to the frequent changed-files sequences.

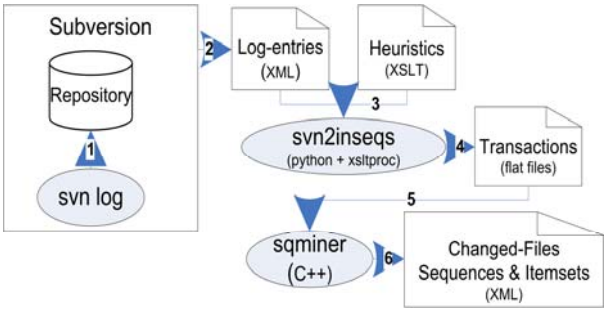

Figure 2. Tool-Chain for Mining Frequent Changed-files

5. MINING TOOLSET

We developed *svn2inseqs* and *sqminer* to process Subversion log-entries and mine the sequences of changed-file. The tool-chain is shown in Figure 2. The overall process is: 1) use the *svn log* command to produce the log-entries in XML format, 2) use *svn2inseqs* to apply a grouping heuristic on the log-entries and obtain the input transactions and events, and 3) finally, use *sqminer* to find the sequences of changed-files. In the following sections, we expand on phases 2 and 3.

5.1. Log-entries to Input-Transactions

Six grouping heuristics, *date (time interval)*, *author*[2] *(committer)*, *file*, *author-date*, *author-file*, and *date-file* are implemented as *XSLT* programs. The *Python* script *svn2inseqs* takes as input the log-entries such as the one shown in Figure 1 and a *XSLT* grouping heuristic, and transforms the log-entries to the corresponding input transactions and events in a flat-file format. One such example of input transactions obtained from the log-entries of the *kdelibs* repository is shown in Figure 3. Here, the log-entries are grouped by the heuristic author-date. The input-transactions file contains a set of events, each specified on a separate line. An event description consists of a generated input transaction-id (e.g., *14*). The transaction-id corresponds to an author-date combination i.e., the change-sets performed by an author (*giessl*) on a particular date (*2005-07-26*). The revision number of a log-entry (i.e., a change-set) is used as an event-id (e.g., *438962* and *438971*). Finally, the files involved in a change-set are listed.

```
...........
14  438962 plastik/plastik.cpp
14  438971 plastik.cpp
...
116  449301 plastik.cpp
116  449436 kstyle.cpp kstyle.h
116  449437 plastik.cpp
116  449483 kstyle.cpp kstyle.h
116  449484 plastik.cpp
116  449494 plastik.cpp
116  449521 plastik.cpp
.......
```

Figure 3. A Snippet of Input Transactions from *kdelibs* Log-entries grouped by Heuristic Author-Date.

5.2. Mining Changed-file Sequences from Input Transactions via *sqminer*

The transactions constructed by the application of heuristics such as the one shown in Figure 3 are fed to the sequence mining tool, *sqminer*. The tool *sqminer* is realized based on the Sequential Patten Discovery Algorithm (SPADE) [19]. The SPADE algorithm utilizes an efficient enumeration of sequences based on common-prefix subsequences and division of search space using equivalence classes. Additionally, it utilizes a vertical input-transaction format for an efficient counting of support values. The configuration parameters of *sqminer* include support, max number of items allowed in a sequence, mining of sequence (association) rules, and output in both the flat-file and XML format.

```
<frequent-sequences input="kdelibs-authordate">
<frequent-sequence size="2" support="6">
  <elements>
    <element temporal-position="1">
      <items><item>plastik.cpp</item></items>
    </element>
    <element temporal-position="2">
       <items><item>plastik.cpp</item></items>
    </element>
  </elements>
  <idpairs>
    <idpair seqid="14" eventid="438971"/>
    <idpair seqid="116" eventid="449437"/>
    <idpair seqid="116" eventid="449484"/>
  </idpairs>
</frequent-sequence>
</frequent-sequences>
```

Figure 4. A Snippet of Sequences of Changed-files from *kdelibs* Log-entries grouped by Heuristic Author-Date.

Figure 4 shows an example of a sequence of changed-paths mined from *kdelib* with a minimum-support value of 3. In this case, a file (*plastik.cpp*) was changed twice in a sequence. Transactions formed from (possibly different) author-date values that have common files in their change-set(s), contribute to the support value of a sequence of changed-files. In Figure 3, transactions *14* and *116* are not formed from the same author-date values but both support the sequence $\{plastik.cpp\} \rightarrow \{plastik.cpp\}$. In addition to the number of elements (i.e., *size*) and support values, information about the transactions (*seqid* and *eventid*) in which these

[2] We do not make the distinction between authors, contributors, volunteers, and so forth.

sequences are found (i.e., *idpairs*) is also stored. This provides the user (application or human) an additional context for their tasks.

Once we have a sequence, it can easily be reduced to an itemset. For example, if a sequences a→b and b→a are found to have the same support (and/or same *id-pairs*), it can be generalized to an itemset {a, b}. The relationship between itemsets and sequences is one-to-many i.e., it is possible that multiple sequences are reduced to a single itemset. For example, the sequence shown in Figure 4 is reduced to itemset as shown in Figure 5. An itemset is a single element sequence composing of all the items in the corresponding sequence. Notice that our approach produces a multi-set. This information not only gives the items that were involved but also the number of their instances implicitly.

6. EVALUATION

The presented heuristics and mining technique are evaluated on an open source system. The primary interest is to show that our approach is able to find frequent sequences of changed-files.

6.1. Dataset Acquisition

The considered version history from the *KDE Subversion* repository is shown in Table 1. The dataset was collected on the 25th of January 2006. The log-entries were collected separately for each of the *KDE* modules and the entire *KDE*. The *svn log* command was used to extract the logs in *XML* format. Table 1 shows the number of revisions, days these revisions were committed, authors, the number of (unique) files involved in the change-sets (i.e., changed files column), and the number of changes performed in these files. Note that the cumulative sum of an attribute values may not agree with the corresponding value of *KDE* due to common values across modules.

```
<frequent-sequences input="kdelibs-authordate">
<frequent-sequence size="2" support="6">
  <elements>
    <element temporal-position="1">
       <items>
         <item>plastik.cpp</item>
         <item>plastik.cpp</item>
       </items>
    </element>
  </elements>
  <idpairs>
    <idpair seqid="14" eventid="438971"/>
    <idpair seqid="116" eventid="449437"/>
  </idpairs>
</frequent-sequence>
</frequent-sequences>
```

Figure 5. A Snippet of Itemsets of Changed-files from kdelibs Log-entries grouped by Heuristic Author-Date.

6.2. Application of Heuristics

After acquiring the dataset, heuristics were applied (*Day, Author*, *File, Author-date*, *Author-file*, and *Day-file*) with the help of a tool *svn2inseqs* and the input files required by *sqminer* were generated. A calendar day is mapped to the heuristic *Time-interval*. In this study, log-entries consisting of more than ten files were pruned. This was done to discard noisy change-sets such as those updating the license information. Transactions with a single event were also ignored as there is no temporal ordering found in a singleton transaction.

The number of transactions and events obtained from the application of each of the heuristics is given in

Table 5. The transactions and events maintain the invariant |events| ≤ |Revisions| on account of pruning. The events are distributed among the transactions based on the grouping dictated by the heuristics. Therefore, the *transaction density* (i.e., number of events in a transaction) is directly dependent on the applied grouping heuristic. The above invariant implies that more the number of transactions, lower the transaction density. On the other end less the number of transactions, higher the transaction density. The maximum changed-files sequence size (i.e., number of elements) is less than or equal to the maximum number of events found in a transaction. Based on the above properties, we have the following observations: 1) transactions with high-density values are likely to find long-size sequences of changed-files with a less number of supporting transactions and 2) transactions with low-density values are likely to find small-size sequences of changed-files with a more number of supporting transactions.

Table 1. Log Information of the *KDE* Source-Code Repository

Modules	Period: (07-25-2005 to 01-25-2006)				
	Revisions	Days	Authors	Changed-Files	Changes in files
arts	3	2	2	3	4
kde-common	295	125	36	30	325
kdeaccesibility	164	71	12	3129	3684
kdeaddons	155	77	19	3085	3623
kdeadmin	91	53	13	2979	3396
kdeartwork	95	32	10	4423	4598
kdebase	1493	173	90	6027	13364
kdebindings	129	43	6	3214	3424
kdeedu	886	154	37	4360	8077
kdegames	216	71	21	3490	4444
kdegraphics	509	147	22	4032	6452
kdelibs	3557	184	124	5262	20604
kdemultimedia	185	84	24	3210	4493
kdenetwork	553	132	31	4860	9012
kdepim	1944	181	57	7522	15990
kdesdk	686	157	32	4178	6727
kdetoys	86	46	16	2917	3193
kdeutils	333	109	31	3947	5858
kdevelop	375	57	15	6965	9085
kdewebdev	19	16	6	2920	2950
KDE	11170	185	230	30648	82662

6.3. Mining Sequences of Changed-Files

sqminer was executed on the transactions of the KDE modules listed in Table 5. The mining of sequences of changed-files was performed on a number of support values. The maximum number of files (i.e., items) in a sequence was set to 20 in all the runs. These support values were selected taking into account the two observations (transaction density) mentioned in the previous section (6.2). Here, the discussion and analysis of the results are limited to a subset of the results. Our intention is to show the distribution of the sequences found in the context of various tool configurations.

Table 6 shows the sequences of changed-files found from the transactions corresponding to the heuristics used in the mining process. The number of sequences (S) found with a configuration

of the minimum support (σ_{min}), the maximum sequence size ($|E_m|$) along with the maximum number of files ($|\alpha_m|$), and the run-time (T) are presented. Sequences were found for a range of minimum support values (σ_{min}). The heuristics Day and Day-Author found sequences in all the cases. The heuristic Author did not report sequences in three cases. The heuristic File and Author-File did not report sequences in six cases. Finally, the heuristic Day-File did not report any sequences in seven cases. There were no sequences found for *arts, kdewebdev*, and *kdeartwork*.

Table 2. Sequences in *kdelibs*

kdelibs	Sequences with no of Elements									
	1	**2**	**3s**	**4**	**5**	**6**	**7**	**8**	**|S|**	**$\|\alpha_m\|$**
Day (3)	900	1304	593	58	1	-	-	-	2856	7
Author(3)	469	2079	3866	4401	2919	1214	272	18	15232	8
File (15)	219	98	23	-	-	-	-	-	340	5
Day-Author(3)	717	274	36	-	-	-	-	-	1027	5
Day-File (15)	44	26		-	-	-	-	-	70	4
Author-File (15)	157	100	10	-	-	-	-	-	267	5

Table 3. Comparison of Itemsets and Sequences in *kdelibs*

	Itemset		**Sequences**		**Ratio**
kdelibs	**|I|**	**$\|\alpha_m\|$**	**|S|**	**$\|\alpha_m\|$**	**|S|/|I|**
Day (3)	2193	7	2856	7	1.3
Author(3)	9575	8	15232	8	1.6
File (15)	263	5	340	5	1.3
Day-Author(3)	907	5	1027	5	1.13
Day-File (15)	55	4	70	4	1.27
Author-File (15)	190	5	267	5	1.41

The results presented in Table 6 shows that sequences of changed-files are found from the log-entries using our approach. However, it remains to be seen that the supplementary information in sequences is useful. To facilitate the discussion, *kdelibs* is taken as a representative. Table 2 shows the sequences of changed-files found by applying each of the six heuristics and the min-support value (e.g., 3). In case of the heuristic Author, more than 15,000 sequences were reported. Further, the distribution of these sequences based on the number of elements it contains is also shown. The maximum number of elements found in sequences was reported to be 8. The maximum number of files found in sequences was also reported to be 8.

These results give interesting insights into the software evolution process used by *kdelib* authors. The long sequences of changed-files under the heuristic Author indicate that related changes are committed in small increments spreading across multiple change-sets (revisions). Furthermore, examining the results under the heuristics Day-Author, the sequences are fewer and relatively smaller in size. This means that related changes are typically not committed on the same day and by the same author. This information leads to the hypothesis that related changes for a high-level modification (e.g., feature or bug-fix) are performed in incremental steps. Even if this hypothesis is not verified, we at least have the historical dependencies between high-level changes.

Moreover, the importance of ordering can be seen for tasks such as assisting a developer with the software-change process. For example, if a sequence consisting of eight elements is treated as an itemset, number of candidates of the combinatorial order may need to be considered. This may lead to lack of precision as many of these combinations may turn-out to be false-positives.

Based on the prior discussion, we provided a case for sequences giving more concise information for tasks such as software-change prediction. However, there is a possibility of a large number of sequences (combinatorial order) being produced compared to itemsets. Our approach facilitates decision-making on this issue. The sequences are also reduced to itemsets and output by *sqminer*. Table 3 facilitates the comparison of sequences and itemsets of changed-files for *kdelibs*. In this case, the number of sequences is less than twice the number of itemsets. Each itemset is essentially ordered. Therefore, our approach serves both the generalization (itemsets) and specialization (sequences) cases.

Table 4. Example of Sequences of Changed-Files in *kdelibs*

kdelibs	**Sequences**
Day (3)	{range.h} → {katedocument.cpp , katedocument.h}
	{KDE4PORTING.html } → { kstringhandler.cpp, kstringhandler.h }
Author (3)	{generic.py} → {openssl.py}
Day-Author (3)	{kstyle.h} → {plastik.cpp} → {kstyle.cpp}
	{ kateregression.cpp } → { kateregression.cpp } → { range.cpp }

We conclude this section, with examples of sequences found in *kdelibs* as shown in Table 4. The sequence shown in the heuristics *Day* is partially ordered. The file *range.h* changed before *katedocument.cpp* and *kdatedocument.h.* However, the order of changes between *katedocument.cpp* and *kdatedocument.*h is undecided. The sequence in the *Day-Author* demonstrates the ordering between an interface file (*kstyle.h*) and an implementation file (*kstyle.cpp*).

7. RELATED WORK

We briefly discuss approaches utilizing information found in source-code repositories maintained by tools such as *CVS* and *Subversion* with a focus on software changes.

Zimmerman et al [20, 21] used *CVS* logs for detecting evolutionary coupling between source-code entities. They employed sliding window heuristics to estimate the atomic commits (change-sets). Association-rules based on itemset mining were formed from the change-sets and used for change-prediction. Yang et al [18] used a similar technique for identifying files that frequently change together. Gall et al [8] used window-based heuristics on *CVS* logs for uncovering logical couplings and change patterns, and German et al [9] for studying characteristics of different types of changes. Hassan et al [11] analyzed *CVS* logs for software-change prediction.

Van Rysselberghe et al [16] utilized *CVS* logs in their approach to find frequently applied changes and presented a 2D visualization technique to help recognize change-relevant information [17]. Bieman et al [3] used logs from software repositories to assist in

the computation of metrics for detecting change-prone classes. Burch et al [4] presented a tool that supports visualization of association rules and sequence rules. However, a very little information is provided on how *CVS* transactions are processed and sequences are mined. Beyer et al [2].used the log information in visualizing clusters of frequently occurring co-changes. Dinh-Trong et al [6] used *CVS* logs for validating previously developed hypotheses on successful open source development. Chen et al [5] incorporated the *CVS* commit messages in their source-code search tool. El-Ramly et al [7] used sequence mining to detect patterns of user activities from the system-user interaction data.

Table 5. Transactions formed by Application of Grouping Heuristics on the Log Information of the *KDE* Source-Code Repository

Modules	(No. of Transactions, No. of Events)					
	Day	Author	File	Day-Author	Day-File	Author-File
arts	(1, 2)	(1,2)	(0, 0)	(1, 2)	(0, 0)	(0, 0)
kde-common	(72, 242)	(8, 267)	(11, 38)	(64, 168)	(5, 10)	(9, 26)
kdeaccesibility	(35 ,111)	(6, 137)	(63, 212)	(31, 93)	(12, 27)	(57, 170)
kdeaddons	(31, 104)	(8, 133)	(52 , 173)	(29, 94)	(18, 36)	(44, 150)
kdeadmin	(18, 51)	(6, 73)	(33, 84)	(16, 46)	(9, 18)	(26, 60)
kdeartwork	(13, 68)	(5, 80)	(17, 40)	(13, 64)	(10, 22)	(15, 35)
kdebase	(157, 1348)	(63, 1337)	(484, 1501)	(235, 1049)	(119, 256)	(349, 879)
kdebindings	(21, 94)	(4, 111)	(17, 121)	(21, 86)	(18, 64)	(20, 118)
kdeedu	(123, 752)	(25, 760)	(344, 1371)	(164, 626)	(168, 424)	(331, 1104)
kdegames	(32, 156)	(9, 178)	(54, 154)	(30, 139)	(8, 16)	(46, 104)
kdegraphics	(109, 427)	(14, 453)	(131, 520)	(96, 335)	(41, 95)	(122, 429)
kdelibs	(182, 3304)	(89, 3271)	(911, 3757)	(605, 2720)	(420, 977)	(719, 2281)
kdemultimedia	(37, 116)	(13, 146)	(55, 175)	(37, 100)	(8, 16)	(54, 128)
kdenetwork	(95, 456)	(25, 482)	(194, 648)	(104, 390)	(70, 146)	(152, 464)
kdepim	(166, 1668)	(46, 1671)	(686, 2186)	(330, 1365)	(203, 444)	(557, 1563)
kdesdk	(125, 618)	(19, 637)	(145, 486)	(141, 517)	(53, 121)	(132, 427)
kdetoys	(20, 58)	(6, 70)	(42, 132)	(17, 49)	(14, 28)	(36, 115)
kdeutils	(63, 251)	(22, 281)	(126, 437)	(66, 212)	(55, 126)	(106, 344)
kdevelop	(28, 294)	(9, 309)	(177, 713)	(32, 282)	(170, 592)	(170, 676)
kdewebdev	(2, 4)	(3, 12)	(6, 12)	(2, 4)	(0, 0)	(0, 0)
KDE	(185, 10092)	(183, 10047)	(3373, 12142)	(1659,8753)	(1274, 2954)	(2773, 8496)

Table 6. Sequences found from the Log Information of the *KDE* Source-Code Repository

Modules	|S|– No. of Sequences, $\|E_m\|$ ($\|\alpha_m\|$)- Max Element-Size (Max) Item-size, and σ_{min}. used in Mining, T – Run-time in Seconds																	
	Day			Author			File			Day-Author			Day-File			Author-File		
	|S| (σmin)	$\|E_m\|$	T.	|S| (σmin)	$\|E_m\|$	T.	|S| (σmin)	$\|E_m\|$	T.	|S| (σmin)	$\|E_m\|$	T.	|S| (σmin)	$\|E_m\|$	T.	|S| (σmin)	$\|E_m\|$	T.
kde-common	30(3)	3(3)	<1	223(3)	6(6)	6	0	-	-	15(3)	2(2)	<1	0	-	-	0	-	-
kdeaccesibility	22(3)	2(2)	<1	1(3)	1(1)	<1	44(10)	1(3)	<1	12(3)	2(2)	<1	0	-	-	19(10)	1(2)	<1
kdeaddons	14(3)	1(7)	<1	1(3)	1(1)	<1	-	-		137(3)	1(7)	<1	31(10)	1(5)	<1	0	-	-
kdeadmin	5(3)	1(1)	<1	0	-	-	1(10)	1(1)	<1	5(3)	1(1)	<1	0	-	-	0	-	-
kdebase	309(3)	2(2)	8	216(3)	3(4)	35	3(25)	1(1)	<1	193(3)	2(2)	1	1(10)	1(1)	<1	0	-	-
kdebindings	31(3)	2(3)	<1	0	-	-	0	-	-	27(3)	2(3)	<1	4(10)	1(1)	<1	2(10)	1(1)	<1
kdeedu	518(3)	3(4)	3	1152(3)	5(6)	7	11(25)	2(2)	<1	390(3)	3(3)	1	4(25)	1(1)	<1	11(25)	2(2)	<1
kdegames	12(3)	2(2)	<1	6(3)	2(2)	<1	5(10)	1(2)	<1	8(3)	2(2)	<1	0	-	-	1(10)	1(1)	<1
kdegraphics	138(3)	3(3)	<1	55(3)	3(4)	1	0	-	-	103(3)	2(4)	<1	4(10)	1(1)	<1	1(25)	1(1)	<1
kdelibs	2856(3)	5(5)	427	15232(3)	8(8)	2247	39(25)	2(2)	10	1027(3)	3(4)	23	19(25)	2(3)	<1	36(25)	2(2)	2
kdemultimedia	19(3)	1(2)	<1	35(3)	4(4)	<1	17(10)	1(3)	<1	17(3)	1(1)	<1	0	-	-	23(10)	1(4)	<1
kdenetwork	215(3)	2(5)	1	67(3)	2(2)	<1	0	-	-	186(3)	2(5)	1	163(10)	2(7)	<1	0	-	-
kdepim	770(3)	3(3)	17	749(3)	6(7)	20	7(25)	1(1)	<1	634(3)	2(4)	3	0	-	-	4(25)	1(1)	<1
kdesdk	102(3)	3(3)	<1	12(3)	2(3)	<1	0	-	-	79(3)	3(3)	<1	2(10)	1(1)	<1	0	-	-
kdetoys	9(3)	1(2)	<1	0	-	-	1528(10)	2(9)	2	7(3)	1(1)	<1	0	-	-	1527(10)	2(9)	4
kdeutils	87(3)	3(3)	<1	20(3)	2(2)	<1	1(25)	1(1)	<1	70(3)	3(3)	<1	35(10)	2(3)	<1	75(10)	3(4)	1
kdevelop	88(3)	3(4)	<1	2(3)	1(1)	<1	2(25)	2(2)	1	82(3)	3(4)	<1	2(25)	2(2)	1	2(25)	2(2)	1
KDE	119(10)	2(3)	13	14(10)	2(2)	2	61(25)	2(2)	15	4301(3)	3(4)	350	23(25)	2(3)	1	52(25)	2(2)	4

8. CONCLUSIONS AND FUTURE WORK

We investigated the problem of mining ordered sequences of changed-files from the change-sets found in source-code repositories. Six heuristics were examined to form input transactions with ordered change-sets. A toolset was developed to uncover the sequences of changed-files from the change-sets. A case-study on the *KDE* project shows that our approach is able to find ordered sequences of changed-files ranging from none to thousands. In our experience, the number of sequences is found to be closely bounded to the number of itemsets. We formed a hypothesis that sequences can be used to better predict and analyze the software evolutionary process.

In future, we plan to assess the effectiveness of the sequences of changed-files for the task of software-change prediction. Sequences of source-code entities such as classes, methods, statements, and expressions will be analyzed and compared. Also, we plan to integrate grouping heuristics based on the textual contents of comments in the change-sets and other repositories (e.g., *Bugzilla*).

9. REFERENCES

[1] Agrawal, R. and Srikant, R. Mining Sequential Patterns in Proceedings of Eleventh International Conference on Data Engineering (Taipei, Taiwan, March, 1995).

[2] Beyer, D. and Noack, A. Clustering Software Artifacts Based on Frequent Common Changes in Proceedings of 13th International Workshop on Program Comprehension (IWPC'05) (St. Louis, Missouri, USA, May 15-16, 2005), 259-268.

[3] Bieman, J. M., Andrews, A. A., and Yang, H. J. Understanding Change-Proneness in OO Software Through Visualization in Proceedings of 11th IEEE International Workshop on Program Comprehension (IWPC'03) (2003), 44-53.

[4] Burch, M., Diehl, S., and Weißgerber, P. Visual Data Mining in Software Archives in Proceedings of Proceedings of the 2005 ACM symposium on Software visualization (St. Louis, Missouri, May 14-15, 2005), 37-46.

[5] Chen, A., Chou, E., Wong, J., Yao, A. Y., Zhang, Q., Zhang, S., and Michail, A. CVSSearch: Searching through Source Code using CVS Comments in Proceedings of Proceedings IEEE International Conference on Software Maintenance (ICSM'01) (2001), 364-373.

[6] Dinh-Trong, T. T. and Bieman, J. M. The FreeBSD Project: a Replication Case Study of Open Source Development. IEEE Transactions on Software Engineering, 31, 6 (2005), 481-494.

[7] El-Ramly, M. and Stroulia, E. Mining Software Usage Data in Proceedings of International Workshop on Mining Software Repositories (MSR'04) (2004), 64-8.

[8] Gall, H., Hajek, K., and Jazayeri, M. Detection of Logical Coupling based on Product Release History in Proceedings of International Conference on Software Maintenance (ICSM'98) (1998), 190-199.

[9] German, D. M. An Empirical Study of Fine-Grained Software Modifications in Proceedings of 20th IEEE International Conference on Software Maintenance (ICSM'04) (2004), 316-25.

[10] German, D. M. Mining CVS Repositories, the SoftChange Experience in Proceedings of International Workshop on Mining Software Repositories (MSR'04) (2004), 17-21.

[11] Hassan, A. E. and Holt, R. C. Predicting Change Propagation in Software Systems in Proceedings of 20th IEEE International Conference on Software Maintenance (ICSM'04) (2004), 284-93.

[12] Huang, S.-K. and Liu, K.-m. Mining Version Histories to Verify the Learning Process of Legitimate Peripheral Participants in Proceedings of International Workshop on Mining Software Repositories (MSR'05) (St. Louis, Missouri, May 17, 2005), 84-78.

[13] Lopez-Fernandez, L., Robles, G., and Gonzalez-Barahona, J. M. Applying Social Network Analysis to the Information in CVS Repositories in Proceedings of International Workshop on Mining Software Repositories (MSR'04) (May 25, 2004), 101-105.

[14] Mockus, A., Fielding, T., and Herbsleb, D. Two Case Studies of Open Source Software Development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology (TOSEM), 11, 3 (July 2002 2002), 309-346.

[15] Tu, Q. and Godfrey, M. W. An Integrated Approach for Studying Architectural Evolution in Proceedings of 10th International Workshop on Program Comprehension (IWPC'02) (2002), 127-136.

[16] Van Rysselberghe, F. and Demeyer, S. Mining Version Control Systems for FACs (Frequently Applied Changes) in Proceedings of International Workshop on Mining Software Repositories (MSR'04) (May 25, 2004), 48-52.

[17] Van Rysselberghe, F. and Demeyer, S. Studying Software Evolution Information By Visualizing the Change History in Proceedings of 20th IEEE International Conference on Software Maintenance (2004), 328-37.

[18] Ying, A. T. T., Murphy, G. C., Ng, R., and Chu-Carroll, M. C. Predicting Source Code Changes by Mining Change History. IEEE Transactions on Software Engineering, 30, 9 (September 2004), 574 - 586.

[19] Zaki, M. J. SPADE: An Efficient Algorithm for Mining Frequent Sequences. Machine Learning, 42, 1-2 (January 2001), 31 - 60.

[20] Zimmermann, T., Weibgerber, P., Diehl, S., and Zeller, A. Mining version histories to guide software changes in Proceedings of 26th International Conference on Software Engineering (2004), 563-72.

[21] Zimmermann, T., Zeller, A., Weissgerber, P., and Diehl, S. Mining Version Histories to Guide Software Changes. IEEE Transactions on Software Engineering, 31, 6 (2005), 429-445.