International Journal of Management, IT & Engineering
Vol. 8 Issue 7, July 2018,
ISSN: 2249-0558 Impact Factor: 7.119
Journal Homepage: http://www.ijmra.us, Email: editorijmie@gmail.com
Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gage as well as in Cabell's Directories of Publishing Opportunities, U.S.A

ASSESSING APPLICABILITY OF SOFTWARE EVOLUTION BASED ASPECT MINING APPROACH

Yasmin Shaikh[*]

Sanjay Tanwani[**]

Abstract

Aspect mining is the search of candidate aspects in existing systems and isolating them from the system into separately described aspects. The crosscutting concerns are candidate aspects. Software Evolution Aspect Mining (SEAM) mine candidate aspects from version history files [1]. In order to assess the applicability of the SEAM in aspect mining and validate the proposed algorithms, version histories of legacy systems are identified for further investigation. In this paper, SEAM is applied on the version histories of two open source software namely, JHotDraw and Weka written in Java. The simple and complex aspect candidate aspects are identified for both the systems and top ranked candidate aspects are listed in the paper.

Keywords:

Keywords: Aspect mining;
data mining;
version history mining;
software evolution;
cross-cutting concern;
frequent pattern mining.

[*] **Assistant Profeesor, International Institute of Professional Studies, Devi Ahilya University, Takshshila Campus, Indore (M. P.), India**

[**] **Professor & Head, School of Computer Science & IT, Devi Ahilya University, Takshshila Campus, Indore (M. P.), India**

1. Introduction

SEAM involves identifying possible aspect code modules by mining version history files [1]. The idea behind this approach is to analyze the history files to identify the files that have been changed together frequently during the evolution of software. The files in frequent patterns are visualized for structural relationship between them. The related files are then recommended as candidate aspects. Two types of candidate aspects are reported – simple candidate aspects and complex candidate aspects. A simple candidate aspect is a set of strongly change coupled files with structural relationship between them. It is evident from the literature available on aspects that crosscutting functionality cut across several files. Combining simple candidate aspects and then obtaining structural relationship among them can find such candidate aspects.

In this paper, the method of data collection from the version archives of both the systems is described. A brief introduction is included about the tools used for performing experimentation. The tools that are used for experimentation include SPMF, CodePro Analytix, and FINT. SPMF is used for frequent pattern mining. CodePro Analytix is used for visualizing structural relationships. FINT is used as benchmarking tool for calculating the precision of experimental results produced by SEAM. The results of frequent pattern mining are presented. The change prone files identified from both the systems are listed. The change coupling visualizations for the largest frequent patterns are presented. The top ranked candidate aspects are also listed for both the systems.

2. Research Method

For thorough evaluation of SEAM, two open source projects JhotDraw and Weka are selected and mined for crosscutting concerns in them. JhotDraw is selected as it is frequently used as an aspect mining benchmark. The version history of JHotDraw is made available for research as Concurrent Versions Systems (CVS) version archive [2]. CVS is a version control system used by Open Source System (OSS) projects Weka is selected, as it is open source software with rich development history [3]. Also, the version history of both the projects is maintained using different versioning system namely CVS and Apache Subversion (SVN). SVN is widely used as revision control system by open source software development community. It overcomes the problems associated with CVS such as commit conflicts and loss of information about renamed or deleted files. Also, the migration from CVS data into SVN repository is easy using tool such as cvs2svn The technique is evaluated for both types of versioning systems.

Table 1 shows few metrics about both the projects stated above. It includes the number of distinct ".java" files present in the system, number of developers involved in the development and maintenance of the system, the number of transactions extracted from version history, date of first and last transactions in the version history. The data comprised for both the system involve same period of time (i.e. around five years) and number of transactions are also similar.

	JhotDraw	Weka
Java Files	504	885
Developers	9	6
Transactions	2698	2015
First Transaction	12-Oct-2000	17-Jul-2008
Last Transaction	25-Apr-2005	18-Jan-2013

Table 1: Transaction statistics from CVS of JHotDraw and SVN of Weka.

To analyze version archives, the data need to be preprocessed. Preprocessing directly affect the quality of result. In SEAM, the data is extracted from version archives. In the present study, CVS repository for JHotDraw available as a CVS dataset is considered [2]. The version files (java, v) are downloaded from the source. Each ",v" file contain whole history of revisions over the corresponding ".java" file along with the code of current release. For Weka, the change history from the hosting website is downloaded [4]. The study of version history files of the software from stable version 3.6.0 through stable version 3.6.9 is done. The history of Weka is maintained using Apache subversion (SVN). SEAM takes into account only revisions of the file not the source code, so the steps required for preprocessing the version history are described below:

Step I: Collect Data

The CVS repository includes revisions associated with each file whereas SVN includes revisions associated with a version of complete software. Therefore, the methods for collecting data from CVS and SVN approach slightly differ. For JHotDraw, the revision statements from each file are extracted. The revision statements contain version number, timestamp, author, state and branch fields. An example of JHotDraw CVS log-file is depicted in Figure 1. In the version history files of Weka, only revision statements are written not the source code. The structure of record in the file contains a record identifier, author who made the change, date & time of change, number of lines changed, path(s) to the files that have been changed, and comment indicating the change. An example of Weka SVN log-file is depicted in Figure 2.

```
head    1.17; access;
symbols
        jhotdraw60b1-release:1.17    package_rename:1.15
        directory_rename:1.14        reorg_6x_split_initial:1.14
        MVC_PHASE1:1.14.0.2          NEW_ATTRIBUTES:1.13.0.2
        DNOYEB1_ALPHA-2:1.11.4.2        release_JHD54b1:1.11
        BUGFIX_670992:1.11.0.6       DNOYEB1_ALPHA-1:1.11.4.2
        dnoyeb1:1.11.0.4        repack:1.11.0.2
        Root_repack:1.11        Before_FigureVisitor:1.10
        JHotDraw_5-3:1.6        JHotDraw_5-2_merged:1.2
        JHotDraw_5-1_initial:1.1        start:1.1.1.1
        vendor:1.1.1; locks; strict;
comment         @# @;
1.17
date    2004.02.01.14.35.11;  author mrfloppy;        state Exp;
branches;
next    1.16;
1.16
date    2004.01.27.16.27.25;  author mtnygard;        state Exp;
branches;
next    1.15;
```

Figure 1: An example of CVS log-file DrawApplet.java from JHotDraw.

```
------------------------------------------------------------------
r6220 | mhall | 2010-01-12 14:37:52 +1300 (Tue, 12 Jan 2010) | 1 line
Changed paths:
   M /branches/stable-3-6/weka/src/main/java/weka/core/version.txt
Updated for 3.6.2
-------------------------------------------------------------------
```

```
r6226 | fracpete | 2010-01-12 18:14:06 +1300 (Tue, 12 Jan 2010) | 1 line
Changed paths:
  M /branches/stable-3-6/weka/src/main/java/weka/core/RelationalLocator.java
  M /branches/stable-3-6/weka/src/main/java/weka/core/StringLocator.java
if src and dest differ in number of attributes/length is different, the error message now states the two different numbers (helpful in figuring out in case this occurs)
```

Figure 2: An example of Weka SVN log-file.

Step II: Data Integration

The revision statements from the version history files of both the projects are extracted and mapped into records. All the revision records are collected into single file. Table 2 and Table 3 shows the structure of records extracted from JHotDraw and Weka projects respectively.

Version	**Date**	**Time**	**Author**	**File**
1.1	2000.10.12	14.57.07	jeckel	applet\DrawApplet.java
1.1	2000.10.12	14.57.08	jeckel	application\DrawApplication.java
1.1	2000.10.12	14.57.08	jeckel	contrib\ChopPolygonConnector.java
1.1	2000.10.12	14.57.08	jeckel	contrib\DiamondFigure.java
1.1	2000.10.12	14.57.08	jeckel	contrib\PolygonFigure.java
1.1	2000.10.12	14.57.08	jeckel	contrib\PolygonHandle.java
1.1	2000.10.12	14.57.08	jeckel	contrib\PolygonScaleHandle.java
1.1	2000.10.12	14.57.08	jeckel	contrib\PolygonTool.java
1.1	2000.10.12	14.57.08	Jeckel	contrib\TriangleFigure.java
1.1	2000.10.12	14.57.08	Jeckel	contrib\TriangleRotationHandle.java

Table 2: Mapping of revision statements from ",v" files of JHotDraw into records.

RecID	Date	Time	Author	File
r4462	7/17/200	9:37:03	mhall	classifiers/functions/Logistic.java

	8			
r4463	7/19/2008	20:40:35	fracpete	filters/unsupervised/attribute/ReplaceMissingValues.java
r4466	7/29/2008	10:29:12	mhall	core/converters/ArffLoader.java
r4466	7/29/2008	10:29:12	mhall	core/converters/C45Loader.java
r4466	7/29/2008	10:29:12	mhall	core/converters/CSVLoader.java
r4466	7/29/2008	10:29:12	mhall	core/converters/LibSVMLoader.java
r4466	7/29/2008	10:29:12	mhall	core/converters/SerializedInstancesLoader.java
r4466	7/29/2008	10:29:12	mhall	core/converters/XRFFLoader.java
r4468	7/30/2008	9:21:27	mhall	classifiers/bayes/DMNBtext.java

Table3: Mapping of revision statement of Weka SVN into records

Definition 1 (Transaction): A transaction T is a pair (t, f). Each pair (t, f) represents the timestamp (t) of the change made to source file f.

To avoid ambiguities, a file is identified with its full signature, including package and class name (if any).

Step III Map co-changing files into transactions

CVS does not keep track of which files have been changed in conjunction in one commit operation whereas SVN includes the set of files that were changed together in one commit. To get the information about files that were changed together from CVS repository, the files are grouped into transaction. To define the time proximity, a time window of one day is selected i.e. the files that were changed on the same date are grouped into transaction. In case of version history data extracted from SVN, the files that are included in single commit are grouped into transaction

Also, to apply frequent pattern mining algorithm, the dataset must be in the form of transaction where the order of transactions determines the temporal relationship. To construct a transaction dataset from revision statements of CVS Algorithm 1 is used.

ALGORITHM 1: MAP CO-CHANGING FILES INTO TRANSACTIONS

Input: *RR*: = set of revision statements having $field_1$ containing date of revision; $field_2$ containing the name of the source file being revised.

RR is sorted on date field.

τ_{max}: = 1 Day //Time window

Output: The transaction database *D*

```
Initialize $d$ := start_date;
           $ld$: = last_date;
$current\_record$: = 1
Read first revision record $r$ from $RR$
do while $d != ld$
    if $field_1$ of $r = d$
        append $field_2$ of $r$ in $D$ at $current\_record$
        read next record
    else
        $d$: = $field_1$ of $r$
        $current\_record$: = $current\_record$ +1
        append $field_2$ of $r$ in $D$ at $current\_record$
        read next record
Output $D$, the transaction database.
```

Given a set of revision statements and size of the window, Algorithm 1 returns a transaction data set by grouping revision records on the basis of time window.

3. Experimantation and Results

3.1 System Configuration

All the experiments are performed on 2.53 GHz intel® CORE™ i3, running 32-bit Windows 7, configured with 3GB RAM. The proposed algorithm in this paper is implemented in Java. The tools that are used for generating frequent patterns (i.e. SPMF) and structural relationships (CodePro) are also implemented in Java.

3.2 Tool for Frequent Pattern Mining

SPMF tool is used for mining frequent patterns using FPGrowth algorithm [5]. SPMF is an open-source data-mining library written in Java for pattern mining. The source code of each algorithm can be integrated in other Java software. Also, SPMF can be used as a standalone program with a simple user interface or from the command line.

The input of FPGrowth is a transaction database (also called as binary context) and a threshold named minsup (minimum support value between 0 and 100 %). In transaction database, an item is represented by a positive integer. A transaction is a line in the text file. In each line (transaction), items are separated by a single space. The output of FPGrowth is a text file with each frequent itemset annotated with its support. The support of an itemset is how many times the itemset appears in the transaction database.

The preprocessed transaction dataset of JhotDraw and Weka includes file names (full path to file) as items. To apply the pattern-mining algorithm using SPMF, the file names are mapped to positive integers. The output is generated in the form of positive integers that are mapped to file names and results are interpreted.

3.3 Tool for Generating Structural Relationship

CodePro Analytix tool is used for visualizing structural relationship among strongly change coupled files [6]. CodePro Analytix is the premier Java software testing tool for Eclipse developers who are concerned about improving software quality and reducing developments costs and schedules. The key features of tool include code analysis, project metrics computation, JUnit test generation, code coverage analysis, similar code analysis and dependency analysis.

In the current research, the dependency analysis is performed using CodePro Analytix. It analyzes and visually depicts the dependencies between projects, packages, and types. Dependencies are displayed in a graphical format. The elements visible at each level of granularity (projects, packages, or types) are displayed as rectangles labeled by both an icon indicating the kind of element being viewed and the name of the element. Dependencies between the elements are displayed as directed lines (lines with arrows at either one end or both). The elements are divided into three groups, which are color coded for ease of recognition. The number of dependencies between nodes is displayed as labels on the connecting lines. If the line is bi-directional, the number of dependencies in both directions will be displayed, separated by a slash ("/").

3.4 Software Evolution based Aspect Mining for JHotDraw

3.4.1 Frequent pattern Generation for JHotDraw

The frequent pattern mining algorithm is applied using SPMF tool to the preprocessed version history dataset of JHotDraw. Table 4 describes the parameters used in algorithm for generating frequent patterns and statistics about the patterns generated at different levels of σ. The first column lists the minimum support percent (σ). The second column presents the number of frequent patterns generated with the specified σ. The third column indicates the number of frequently appearing files that are generated from the extracted frequent patterns. These files are frequent patterns of size one. The fourth column shows the size of maximal frequent set(s) generated with the specified σ.

The value of σ is varied so that the reasonable number of files gets involved in frequent patterns. The value of σ is varied within 10% to 20% range. Good numbers of patterns are generated at 10% threshold value. Very few patterns are generated at 15% and 20% threshold value. The tool failed to generate the results for σ less than 10%.

A moderate value of threshold σ is selected. If the threshold value is kept overly high, very few patterns are generated. If the threshold value is kept very low, then large numbers of patterns are generated. The resulting patterns contain large number of redundant sub-patterns. The precision of results may fall with large set of frequent patterns as only few patterns are actually correct.

Min. Support σ	No. of frequent patterns	No. of files	Size of Largest Pattern
8	--	--	--
10	**481**	**57**	**7**
15	28	19	4
20	9	7	3

Table 4 : Statistics of frequent patterns extracted from JHotDraw and Weka.

3.4.2 Performance Analysis of FP-Growth Algorithm for JHotDraw

The statistics generated by SPMF tool includes the approximate memory and time consumed by the algorithm. Figure 3 shows a bar chart about memory and time consumed by algorithm to generate frequent patterns. The memory consumed by the algorithm is around 52MB to 55MB. The maximum time taken by algorithm is for support percent 10 since most number of patterns are generated at this level. The time taken at 15 and 20 percent support is relatively low. Thus, it can be concluded here that the memory requirement does not vary greatly at these support percentages but time requirements vary with support percentage as number of frequent patterns also vary.

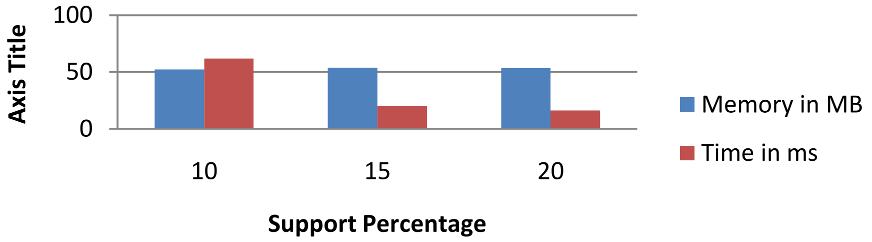


Figure 3: Memory and time taken by FPGrowth algorithm at different support percentages for JHotDraw.

3.4.3 Change Prone Files for JHotDraw

Change prone files are the files that have been changed frequently during the evolution of software. On applying frequent pattern mining algorithm to the version history files, frequent patterns of size 1 to 7 (for JHotDraw) are generated. The frequent patterns of size one include the change prone files. Table 5 shows the change prone files with their support count for JHotDraw.

The experimental results show that in JHotDraw only 16% of total number of files are change prone files and are involved in maintenance more often. Thus, it is concluded that a very less percentage of software code is involved in maintenance activity. The two files that are changed most frequently are **application\DrawApplication.java** and **standard\StandardDrawingView .java**. The class DrawApplication defines a standard presentation for standalone drawing editors. The class StandardDrawingView presents a standard drawing view. Both the classes are related to GUI and are also strongly change coupled as shown in next section.

Change Prone Files	**Support Count**
application\DrawApplication.java	30
standard\StandardDrawingView.java	30
contrib\MDIDesktopPane.java	16
samples\javadraw\JavaDrawApp.java	16
applet\DrawApplet.java	15
figures\TextFigure.java	15
contrib\MDI_DrawApplication.java	14
contrib\GraphicalCompositeFigure.java	13
contrib\TextAreaFigure.java	13
standard\CreationTool.java	13
standard\StandardDrawing.java	13
standard\AbstractTool.java	12
figures\LineConnection.java	11
samples\pert\PertApplication.java	11
standard\ChangeConnectionEndHandle.java	11
util\Geom.java	11
contrib\CustomSelectionTool.java	10
framework\DrawingChangeListener.java	10
standard\DeleteCommand.java	10
standard\CompositeFigure.java	10
standard\ChangeConnectionHandle.java	10
standard\SelectionTool.java	10

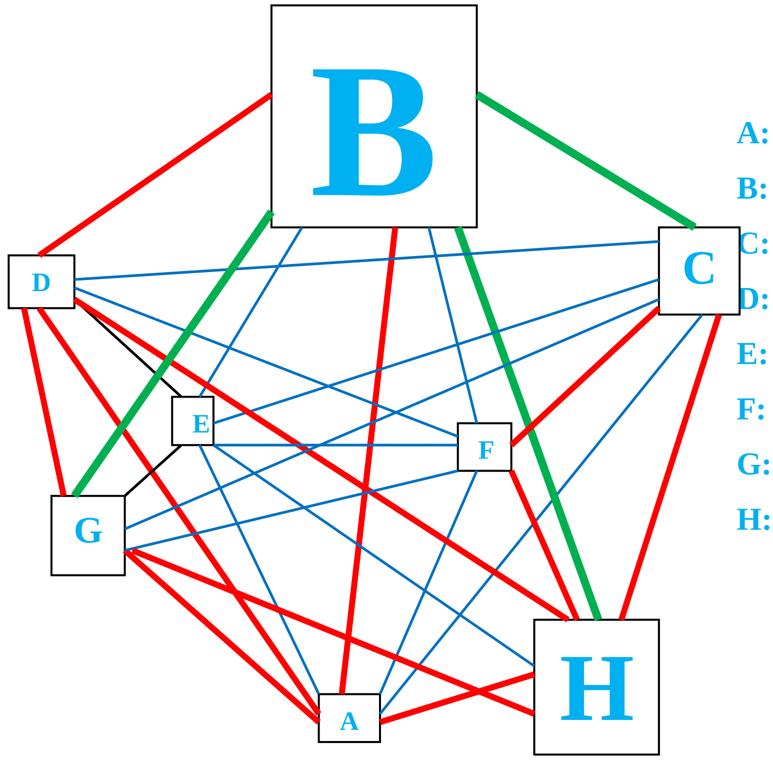
util\StorableInput.java	10
contrib\dnd\DragNDropTool.java	9
contrib\html\HTMLTextAreaFigure.java	9
contrib\MiniMapView.java	9
contrib\zoom\ZoomDrawingView.java	9
figures\ImageFigure.java	9
figures\PolyLineFigure.java	9
framework\Figure.java	9
samples\nothing\NothingApp.java	9
standard\AbstractFigure.java	9
standard\ConnectionTool.java	9
standard\ChangeAttributeCommand.java	9
standard\PasteCommand.java	9
standard\ToolButton.java	9
contrib\JScrollPaneDesktop.java	8
contrib\StandardLayouter.java	8
contrib\TextAreaTool.java	8
figures\AttributeFigure.java	8
figures\GroupFigure.java	8
figures\FigureAttributes.java	8
figures\TextTool.java	8
figures\RoundRectangleFigure.java	8
samples\javadraw\JavaDrawViewer.java	8
samples\net\NetApp.java	8
standard\BringToFrontCommand.java	8
standard\AbstractCommand.java	8
standard\AbstractHandle.java	8
samples\pert\PertDependency.java	8
standard\DecoratorFigure.java	8
standard\CutCommand.java	8
standard\ChangeConnectionStartHandle.java	8

standard\FigureTransferCommand.java	8
standard\SelectAreaTracker.java	8
standard\QuadTree.java	8
util\Iconkit.java	8

Table 5: Change prone files for JHotDraw

3.4.4 Change Coupling Visualization for JHotDraw

Change coupling between files indicates that when one file is changed then the change coupled file is also required to be changed. The frequent patterns of size two and more denote the change coupling between members of pattern. The largest size of frequent patterns generated for JHotDraw is 7. There are 57 change prone files. Thus, it is concluded that very few number of files are associated with change coupling relationship over the period of evolution. The change coupling graphs for largest size pattern for JHoDraw is presented in Figure 4.



Legend

filters/unsupervised/instance/

A:	subsetbyexpression/Scanner.java
B:	gui/AttributeListPanel.java
C:	gui/AttributeSummaryPanel.java
D:	gui/beans/AssociatorCustomizer.java
E:	gui/beans/BatchClassifierEvent.java
F:	gui/beans/ClassAssigner.java
G:	gui/beans/ClassifierCustomizer.java
H:	gui/beans/CostBenefitAnalysis.java

Figure 4: Change coupling relationships between files of largest frequent pattern for JHotDraw.

3.4.5 Visualization of Structural Relationships of Strongly Change Coupled Files for JHotDraw

The strongly change coupled files of JHotDraw are visualized for structural relationships among them. The dependency graphs are generated for strongly change coupled files in frequent patterns of size two or more. First the graph is generated for largest pattern. Initially, a high level view of dependency graph is generated at package level for JHotDraw. Figure 5 shows a dependency graph at package level of JHotDraw generated using CodePro . The granules (here packages) are displayed as rectangles with an icon for kind of element. Dependencies between the elements are displayed as directed lines (lines with arrows at either one end or both).

The elements are divided into three groups, which are color coded for ease of recognition. The first group contains internal elements that are directly selected for analysis. By default, internal elements are displayed in black. The second group contains external elements, which are elements that were neither selected nor contained in selected elements, but that are referenced by them. By default, external elements are displayed in gray. The third group is a subset of the internal elements; those that are elements of a strongly change coupled components. By default they are displayed in red. The thickness of each line further indicates whether the dependency is bi-directional or not. The number of dependencies between nodes is displayed as labels on the connecting lines. If the line is bi-directional, the number of dependencies in both directions will be displayed, separated by a slash ("/").

When a dependency analysis is being viewed, double clicking on some portion of the graph can access finer levels of granularity. Double clicking on an element will display a graph containing only those elements within the element that was clicked on and the elements that are dependent on the clicked element. Thus, the structural relationship is analyzed at file level. On the basis of such detailed analysis, candidate aspects are identified.

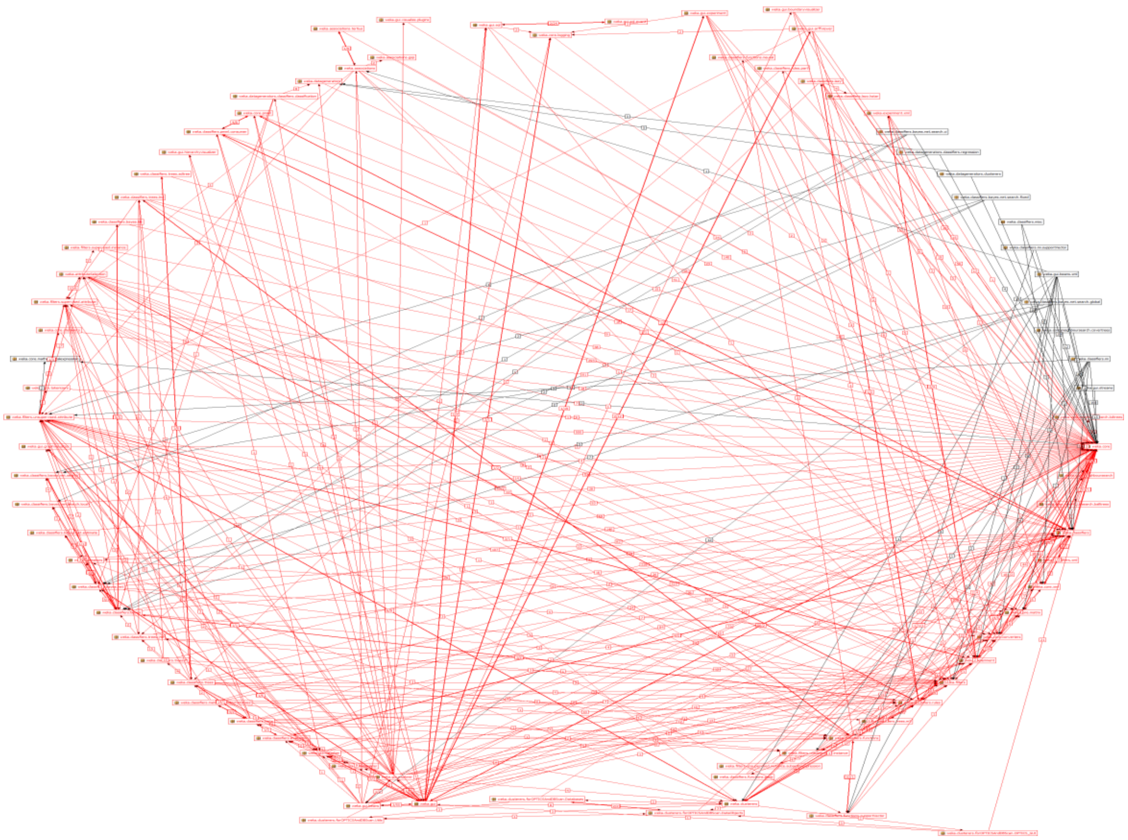


Figure 5: Visualization of structural relationship for JHotDraw using dependency graph generated using CodePro plug-in for Eclipse.

3.4.6 Candidate Aspects for JHotDraw

The frequent patterns are generated using frequent pattern mining algorithm.The strongly changed coupled files and simple candidate aspects are generated using the algorithms described in [1]. These simple candidate aspects are then visualized for structural relationships. The candidate aspects are marked true or false manually on the basis of structural relationship graphs. Manual inspection is required to filter out the redundant concepts and to group together similar concepts. The simple candidates are then combined and complex candidate aspects are generated using algorithm described in [1]. Out of 481 frequent patterns, most of the patterns are redundant, so they got pruned. 26 simple candidate aspects and 14 complex candidate aspects are identified after applying the algorithms for identifying simple and complex candidate aspects. Top 11 candidate aspects are presented in Table 6. Candidate aspect #4 and candidate aspect #6 are complex candidate aspects. The remaining candidate aspects are simple candidate aspects.

Candidate Aspect # 1: Draw and View Figure	**Candidate Aspect # 2: Edit and View Figure**
application\DrawApplication.java figures\TextTool.java framework\Figure.java standard\AbstractFigure.java standard\CompositeFigure.java standard\DecoratorFigure.java standard\StandardDrawingView.java	application\DrawApplication.java standard\AbstractTool.java standard\BringToFrontCommand.java standard\PasteCommand.java standard\StandardDrawingView.java standard\DeleteCommand.java standard\CutCommand.java
Candidate Aspect # 3: Drawing and Viewing Tools	**Candidate Aspect # 4: Manage Changes**
application\DrawApplication.java contrib\MDI_DrawApplication.java standard\AbstractTool.java standard\StandardDrawingView.java	application\DrawApplication.java framework\DrawingChangeListener.java standard\StandardDrawing.java standard\StandardDrawingView.java
Candidate Aspect # 5: Create and Store New Figure	**Candidate Aspect # 6: View Figure Details**
application\DrawApplication.java standard\CreationTool.java standard\StandardDrawingView.java util\StorableInput.java	application\DrawApplication.java standard\SelectionTool.java contrib\CustomSelectionTool.java standard\StandardDrawingView.java
Candidate Aspect # 7: Manage Tool Buttons	**Candidate Aspect # 8: Connect Figure and Text**
application\DrawApplication.java standard\CreationTool.java standard\StandardDrawingView.java standard\ToolButton.java	application\DrawApplication.java figures\LineConnection.java figures\TextFigure.java standard\AbstractFigure.java
Candidate Aspect # 9: Move Figure	standard\ChangeConnectionEndHandle.java
standard\AbstractFigure.java standard\CompositeFigure.java	standard\ChangeConnectionHandle.java standard\ConnectionTool.java

standard\DecoratorFigure.java	**Candidate Aspect # 11: Persistence**
figures\AttributeFigure.java	figures\AttributeFigure.java
figures\ImageFigure.java	figures\ImageFigure.java
figures\GroupFigure.java	figures\GroupFigure.java
figures\FigureAttributes.java	figures\FigureAttributes.java
figures\PolyLineFigure.java	figures\PolyLineFigure.java
figures\RoundRectangleFigure.java	figures\RoundRectangleFigure.java
Candidate Aspect # 10: Draw Geometric Shape	figures\LineConnection.java
application\DrawApplication.java	figures\TextFigure.java
standard\StandardDrawingView.java	standard\AbstractFigure.java
util\Geom.java	standard\DecoratorFigure.java

Table 6: Top ranked candidate aspects for JHotDraw.

3.5 Software Evolution based Aspect Mining for Weka

3.5.1 Frequent pattern Generation for Weka

The frequent pattern mining algorithm is applied to the preprocessed version history dataset of Weka. Table 7 describes the parameters used in algorithm for generating frequent patterns. The first column lists the minimum support (σ) percent. The second column presents the number of frequent patterns generated with the specified σ. The third column indicates the number of frequently appearing files that are generated from the extracted frequent patterns.

These files are frequent patterns of size one. The fourth column contains the size of largest (maximal) frequent pattern. The value of σ is varied so that the reasonable number of files gets involved in frequent patterns. The support is varied within 1% to 3% range since no results are generated above this threshold. Good numbers of results are generated at 1% threshold value.

Min. Support σ	No. of frequent patterns	No. of files	Size of Largest Pattern
0.8	8679	119	14
1	**384**	**68**	**8**
2	14	14	2
3	5	5	2

Table 7: Statistics from frequent patterns extracted for Weka.

3.5.2 Performance Analysis of FP-Growth Algorithm for Weka

Figure 6 shows a bar chart about memory and time consumed by algorithm to generate frequent patterns. The memory consumed by the algorithm is between 3MB to 7MB. The maximum time taken by algorithm is for support percent 0.8 since most number of patterns are generated at this level. The time taken at 1, 2, and 3 percent support is relatively low. Thus, it can be concluded here that the memory requirement does not vary greatly at these support percentages but time requirements vary with support percentage as number of frequent patterns also vary greatly. Also, the time taken is in milliseconds, the frequent patterns are generated in very short period of time.

3.5.3 Change Prone Files for Weka

Change prone files are the files that have been changed frequently during the evolution of software. On applying frequent pattern mining algorithm to the version history files, the frequent patterns of size 1 to 8 are generated.

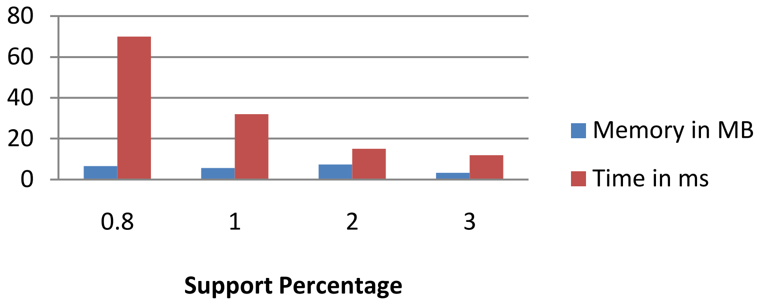


Figure 6: Memory and time taken by FPGrowth algorithm at different support percentages for weka.

The frequent patterns of size one include the change prone files. Table 8 shows the change prone files for Weka. The experimental results show that only 13% of total number of files in Weka are change prone files. Thus, it is concluded that a limited portion of software code is involved in maintenance activity.

The file gui/AttributeListPanel.java is changed most frequently during the evolution of software. This class creates a panel that displays the attributes contained in a set of instances, letting the user select a single attribute for inspection. Before applying any data mining algorithm using

Weka, the user has to select the attributes. Therefore, this class is related with all the data mining techniques that are applicable using Weka.

The gui/beans/ CostBenefitAnalysis.java is the second most frequently modified file. This is a Java bean that aids in analyzing cost/benefit tradeoffs. Again, this analysis is performed on application of each algorithm. Also, it is evident from set of frequent patterns that these classes have high change coupling relationship with other change prone classes. Thus, it can be concluded here that the classes that are related to more number of classes are change prone files.

3.5.4 Change Coupling Visualization for Weka

Change coupling between files indicates that when one file is changed then the change coupled file is also required to be changed. The frequent patterns of size two and more denote the change coupling between members of pattern. The largest size of frequent patterns generated for Weka is 8.

Change Prone Files	**Support Count**
gui/AttributeListPanel.java	28
gui/beans/CostBenefitAnalysis.java	17
gui/experiment/GeneratorPropertyIteratorPanel.java	16
gui/beans/ClassValuePicker.java	12
gui/AttributeSummaryPanel.java	11
clusterers/SimpleKMeans.java	10
gui/beans/IncrementalClassifierEvaluatorCustomizer.java	10
gui/explorer/Messages.java	10
gui/experiment/HostListPanel.java	10
gui/beans/CrossValidationFoldMaker.java	9
classifiers/trees/DecisionStump.java	8
core/converters/LibSVMLoader.java	8
gui/beans/ClassifierCustomizer.java	8
gui/experiment/RunPanel.java	8
gui/sql/event/ResultChangedEvent.java	7

gui/sql/ResultSetHelper.java	7
classifiers/bayes/ComplementNaiveBayes.java	7
classifiers/bayes/net/GUI.java	7
clusterers/MakeDensityBasedClusterer.java	7
gui/beans/BeanConnection.java	7
gui/beans/DataVisualizer.java	7
gui/beans/Startable.java	7
attributeSelection/ClassifierSubsetEval.java	6
classifiers/bayes/NaiveBayesMultinomialUpdateable.java	6
classifiers/trees/BFTree.java	6
clusterers/forOPTICSAndDBScan/DataObjects/EuclideanDataObject.java	6
gui/AttributeVisualizationPanel.java	6
gui/beans/ClassAssigner.java	6
gui/beans/StructureProducer.java	6
gui/beans/KnowledgeFlow.java	6
gui/visualize/MatrixPanel.java	6
core/AllJavadoc.java	5
clusterers/forOPTICSAndDBScan/OPTICS_GUI/OPTICS_Visualizer.java	5
filters/unsupervised/attribute/InterquartileRange.java	5
gui/arffviewer/ArffViewerMainPanel.java	5
filters/unsupervised/instance/subsetbyexpression/Scanner.java	5
gui/beans/ClassValuePickerCustomizer.java	5
gui/beans/AssociatorCustomizer.java	5
filters/unsupervised/attribute/NumericTransform.java	5
gui/experiment/ExperimenterDefaults.java	5
gui/experiment/Experimenter.java	5
gui/beans/FlowRunner.java	5
gui/graphvisualizer/Messages.java	5
gui/experiment/SimpleSetupPanel.java	5

gui/explorer/AssociationsPanel.java	5
gui/AttributeSummaryPanel.java	4
gui/sql/InfoPanelCellRenderer.java	4
classifiers/functions/Winnow.java	4
classifiers/rules/Prism.java	4
clusterers/sIB.java	4
filters/unsupervised/attribute/AddNoise.java	4
filters/unsupervised/attribute/NominalToBinary.java	4
experiment/AveragingResultProducer.java	4
experiment/InstanceQuery.java	4
experiment/Experiment.java	4
gui/AttributeSelectionPanel.java	4
filters/unsupervised/instance/SubsetByExpression.java	4
gui/beans/BatchClassifierEvent.java	4
gui/boundaryvisualizer/Messages.java	4
gui/beans/GraphViewer.java	4
gui/beans/InstanceStreamToBatchMaker.java	4
gui/beans/ClustererCustomizer.java	4
gui/beans/CrossValidationFoldMakerCustomizer.java	4
gui/beans/PredictionAppenderCustomizer.java	4
gui/SimpleCLIPanel.java	4
gui/experiment/OutputFormatDialog.java	4
gui/treevisualizer/TreeDisplayEvent.java	4
gui/streams/InstanceLoader.java	4

Table 8: Change prone files for Weka.

Legend

A: application\DrawApplication.java

There are 68 change prone files. Thus, it is concluded that very few number of files are associated with change coupling relationship over the period of evolution. The change coupling graphs for largest size pattern for Weka is shown in Figure 7.

B: figures\TextTool.java
C: framework\Figure.java
D: standard\AbstractFigure.java
E: standard\CompositeFigure.java
F: standard\DecoratorFigure.java
G: standard\StandardDrawingView.java

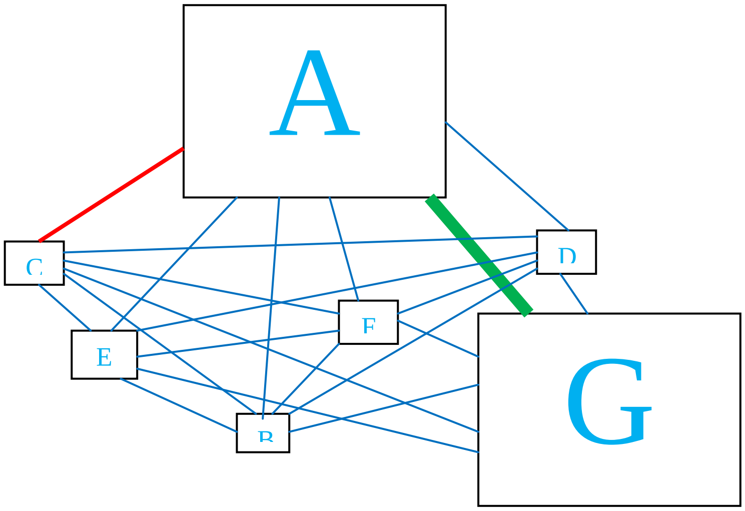


Figure 7: Change coupling relationships for largest frequent pattern for Weka

The change coupling relationship graph for Weka shows that the file B (gui/AttributeListPanel.java) is the file that has been changed the most during the evolution of software. Similarly, the file E (gui/beans/ BatchClassifier Event .java) has been changed least number of times during evolution of software. The files B-C, B-H, and B-G are most strongly change change coupled files. The files A-B, A-D, A-G, A-H, B-D, C-F, C-H, D-G, D-H, F-H, and G-H are moderately change coupled. The remaining file sets are least change coupled files among the set of files.

3.5.5 Visualization of Structural Relationships among Strongly Change Coupled Files for Weka

The strongly change coupled files generated in previous step are visualized for structural relationships among them. The dependency graphs are generated using CodePro tool for strongly change coupled files in frequent patterns of size two or more. First the graph is generated for largest pattern generated for. Similarly, the dependency graph for second largest pattern (and so on) is generated with pruning of proper subsets at each level. Figure 8 shows dependency graph for the largest pattern in Weka at class level.

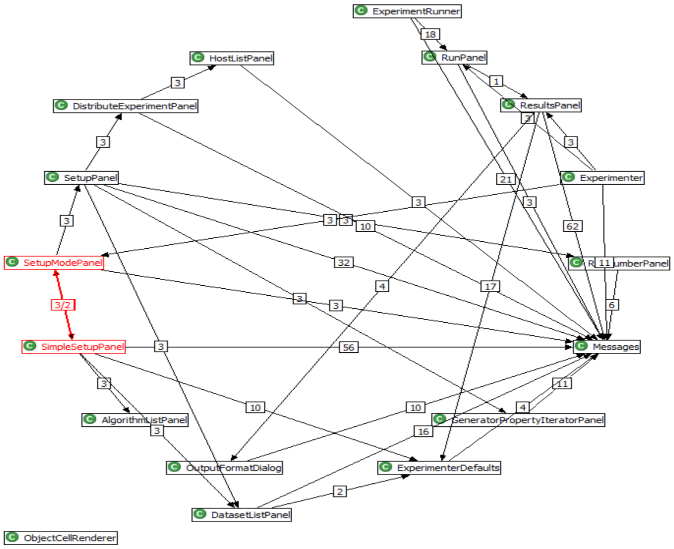


Figure 8: Visualization of structural relationship using dependency graph generated using CodePro plug-in for Eclipse

3.5.6 Candidate Aspects for Weka

The frequent patterns for Weka are generated using frequent pattern mining algorithm. The strongly changed coupled files, simple candidate aspects, and complex candidate aspects are generated similarly as generated for JHotDraw in Section 3.4.6. Out of 384 frequent patterns, most of the patterns are redundant, so they got pruned. 29 simple candidate aspects and 10 complex candidate aspects are identified after applying the algorithms for identifying simple and complex candidate aspects. Top 7 candidate aspects are presented in Table 9. Candidate aspect # 5 is complex candidate aspect and remaining aspects are simple candidate aspects.

Candidate Aspect # 1: Attribute Visualization and Selection	**Candidate Aspect # 2: GUI Customization**
filters/unsupervised/instance/subsetbyexpression/Scanner.java	gui/beans/AssociatorCustomizer.java
gui/AttributeListPanel.java	gui/beans/ClassifierCustomizer.java

gui/AttributeSummaryPanel.java	gui/beans/DataVisualizer.java
gui/beans/ClassAssigner.java	**Candidate Aspect # 4: Experimenter Console Setting**
Candidate Aspect # 3: Message Window	gui/experiment/Experimenter.java
gui/boundaryvisualizer/Messages.java	gui/experiment/ExperimenterDefaults.java
gui/experiment/OutputFormatDialog.java	gui/experiment/GeneratorPropertyIteratorPanel.java
gui/explorer/Messages.java	gui/experiment/HostListPanel.java
gui/graphvisualizer/Messages.java	**Candidate Aspect # 6: Tradeoff Analysis**
Candidate Aspect # 5: Classification Settings	gui/beans/CostBenefitAnalysis.java
gui/AttributeListPanel.java	**Candidate Aspect #7: Subset Evaluator**
gui/AttributeSummaryPanel.java	filters/unsupervised/instance/subsetbyexpression/Scanner.java
gui/beans/CrossValidationFoldMaker.java	
gui/beans/IncrementalClassifierEvaluatorCustomizer.java	
gui/beans/ClassValuePicker.java	

Table 9: Candidate aspects for Weka.

4. Conclusion

A systematic technique to collect data from version archives is proposed in this paper. Also, a detailed data preprocessing approach is introduced. An algorithm is proposed to map the version archive data in the form of transactions. This is especially useful when the user wants to apply different data mining techniques for mining version archives. Change prone files are extracted from version history of software using data mining technique. These change prone files help in identifying the code portions modified most frequently. These files can be refactored to get more maintainable, portable, evolvable, and testable code modules. Co-changing files (change coupled files) are extracted from version history of software. These change-coupling relationships are determined by visualizing frequent patterns on the basis of the support counts of files in the frequent pattern. The results are extremely useful in guiding software maintenance process and enhance maintainability of software. The results produced by SEAm for JHotDraw and Weka shows that the approach can be applied easily to any project with rich development history maintained in the form of CVS or SVN.

References

[1] Shaikh, Y., & Tanwani, S. (2017). Software Evolution-based Aspect Mining: A Novel Approach. *International Journal of Data Mining and Emerging Technologies*, 7(2), 97-106.

[2] M. Monperrus, and M. Martinez, CVS-Vintage: A Dataset of 14 CVS Repositories of Java Software, http://hal.archives-ouvertes.fr/hal-00769121, 2012.

[3] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten, "The WEKA Data Mining Software: An Update*", SIGKDD Explorations*, Vol. 11, no. 1, 2009.

[4] https://www.cs.waikato.ac.nz/~ml/weka/history.html

[5] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Sotlani, C. Wu., and V. Tseng, "SPMF: a Java Open-Source Pattern Mining Library", *Journal of Machine Learning Research (JMLR*), vol. 15, pp. 3389-9993, 2014. http://www.philippe-fournier-viger.com/SPMF/index.php, 2014.

[6] CodePro, Source Code Testing and Visualization Eclipse Plug-in, https://developers.google.com/java-dev-tools/codepro/doc / analytix