

Anvaya: An Algorithm and Case-Study on Improving the Goodness of Software Process Models generated by Mining Event-Log Data in Issue Tracking Systems

Perna Juneja

Computer Science

Indraprastha Institute of Information Technology, Delhi (IIIT-D), India

A Thesis Report submitted in partial fulfilment for the degree of

MTech Computer Science

9 March 2015

1.: Prof. Ashish Sureka (Thesis Adviser)

2. Prof. Vikram Goyal (Internal Examiner)

2. Prof. Y. Raghuram Reddy (External Examiner)

Day of the defense: 9 March 2015

Signature from Post-Graduate Committee (PGC) Chair:

Abstract

Issue Tracking Systems (ITS) such as Bugzilla can be viewed as Process Aware Information Systems (PAIS) generating event-logs during the life-cycle of a bug report. Process Mining consists of mining event logs generated from PAIS for process model discovery, conformance and enhancement. We apply process map discovery techniques to mine event trace data generated from ITS of open source Firefox browser project to generate and study process models. Bug life-cycle consists of diversity and variance. Therefore, the process models generated from the event-logs are spaghetti-like with large number of edges, inter-connections and nodes. Such models are complex to analyse and difficult to comprehend by a process analyst. We improve the Goodness (fitness and structural complexity) of the process models by splitting the event-log into homogeneous subsets by clustering structurally similar traces. We adapt the K-Medoid clustering algorithm with two different distance metrics: Longest Common Sub sequence (LCS) and Dynamic Time Warping (DTW). We evaluate the goodness of the process models generated from the clusters using complexity and fitness metrics. Process models generated after clustering have high degree of fitness and less structural complexity and thus are easier to comprehend compared with the process model generated from the entire event-log. We study back-forth & self-loops, bug reopening, and bottleneck in the clusters obtained and show that clustering enables better analysis. We also propose an algorithm to automate the clustering process -the algorithm takes as input the event log and returns the best cluster set.

I dedicate my MTech Thesis to my father Sunil Kumar Juneja for his never-ending love, support and encouragement. I am blessed to have a father like him.

Acknowledgements

First and foremost I offer my deepest gratitude to my advisor, Dr. Ashish Sureka, who has supported me throughout my thesis. His patience and knowledge provided me the room to work in my own way. I attribute the level of my Masters degree to his encouragement and effort without which this thesis would not have been completed or written. One simply could not wish for a better or friendlier advisor.

Besides my advisor, I would like to deeply thank my esteemed committee members Prof. Y. Raghu Reddy and Prof. Vikram Goyal for agreeing to evaluate my thesis.

My sincere thanks also goes to Divya Kundra for helping me and spending her valuable time to review my thesis. I would also like to thank all my friends at IIIT- Delhi for their encouragement and insightful comments. Last but not the least, I would like to thank all my family members who encouraged and kept me motivated throughout the thesis.

Declaration

This is to certify that the MTech Thesis Report titled **Anvaya: An Algorithm and Case-Study on Improving the Goodness of Software Process Models generated by Mining Event-Log Data in Issue Tracking Systems** submitted by **Prerna Juneja** for the partial fulfillment of the requirements for the degree of *MTech in Computer Science* is a record of the bonafide work carried out by her under my guidance and supervision at Indraprastha Institute of Information Technology, Delhi. This work has not been submitted anywhere else for the reward of any other degree.

Professor Ashish Sureka

Indraprastha Institute of Information Technology, New Delhi

Contents

List of Figures	vi
List of Tables	viii
1 Research Motivation and Aim	1
1.1 Issue Tracking Systems	1
1.2 Process Mining	5
1.3 Problem Motivation, Definition and Aim	11
2 Related Work and Research Contributions	13
2.1 Related Work	13
2.1.1 Problem of Spaghetti Process Models	13
2.1.2 Trace Clustering	14
2.1.3 Conformance Measurement	15
2.2 Novel Research Contributions	15
3 Research Framework and Solution Approach	17
4 Experimental Dataset	23
5 Clustering	26
5.1 K Medoid Clustering	27
5.1.1 Longest Common Subsequence metric	28
5.1.2 Dynamic Time Warping metric	31
6 Evaluation	33
6.1 Complexity	33
6.2 Fitness	34

7	Experimental Results	36
8	Analysis	42
8.1	Self-loop Analysis	42
8.2	Back-Forth Analysis	46
8.3	Event Analysis	47
8.4	Activity Frequency Analysis	47
8.5	Reopen Analysis	51
8.6	Unique Traces	54
8.7	Bottleneck Identification	54
9	Automate Clustering to Determine the Best Cluster Solution	59
10	Limitations and Future Work	62
11	Conclusion	63
	References	64

List of Figures

1.1	Software Development Life Cycle with the Maintenance Phase highlighted (Figure taken from 1)	1
1.3	Snapshot of Mozilla Bug History (Bug ID 239534	4
1.4	Bug Life Cycle in Bugzilla (Figure taken from 1)	5
1.5	Process Mining Types (a) discovery (b) conformance and (d) enhancement (Figure taken from (1))	6
1.6	Snapshot of Disco after opening an Event Log in it. One can select the column and mark it as CaseID, Activity, Timestamp, Resource or Other.	8
1.7	Process Discovery: Process Model generated from the given Event Log using Disco	9
1.8	Problem of Spaghetti Model	10
3.1	Architecture Diagram and Data Processing Pipeline for Anvaya Framework (Clustering-Based Approach for Improving the Goodness of Software Process Models Derived from Event-Logs)	18
3.2	Data Transformation	19
4.1	Case Duration	24
4.2	Number of Cases vs Case Variants	24
4.3	Mozilla Firefox Event Log Data	25
5.1	Clustering	26
5.2	Software Quality Assurance Manager using his Domain Knowledge to Determine the value of K in an Iterative Process	28
5.3	Longest Common Subsequence	30
5.4	Dynamic Time Warping	31

LIST OF FIGURES

7.1	Process model generated using Disco from the entire event log (a) and event log of six clusters (b-g) obtained using LCS distance metric. . . .	40
7.2	Process model generated using Disco from the entire event log (a) and event log of six clusters (b-g) obtained using DTW distance metric. . . .	41
8.1	Self-Loop	42
8.2	Back-Forth	46
8.3	No. of Events per Case in each of the six Clusters.	48
8.4	Distribution of some Frequently Occurring Activities over the cases . . .	51
8.5	Reopen Analysis.	52
8.6	Bottleneck Analysis.	56

List of Tables

3.1	Count and Description for some Activities.	22
4.1	Experimental Dataset Details (Mozilla Firefox Project)	23
7.1	Cluster Description	37
7.2	Complexity and Fitness Metric of the Spaghetti Model Generated from the entire Event Log as well as the Six Clusters Generated by K-medoid Algorithm using LCS as the Distance Metric	39
7.3	Complexity and Fitness Metric of the Spaghetti Model Generated from the entire Event Log as well as the Six Clusters Generated by K-medoid Algorithm using DTW as the Distance Metric	39
8.1	Self Loops and Back-Forth Analysis	45
8.2	Absolute Frequency (Abs) and Relative Frequency Percentage (Rel freq.) for all Activities present in the Clusters.	48
8.3	Most Occurring Variants of Main Model and each of the Six Clusters with their Frequency of Occurrence	55
9.1	Automated Clustering Algorithm Analysis	60

1

Research Motivation and Aim

1.1 Issue Tracking Systems

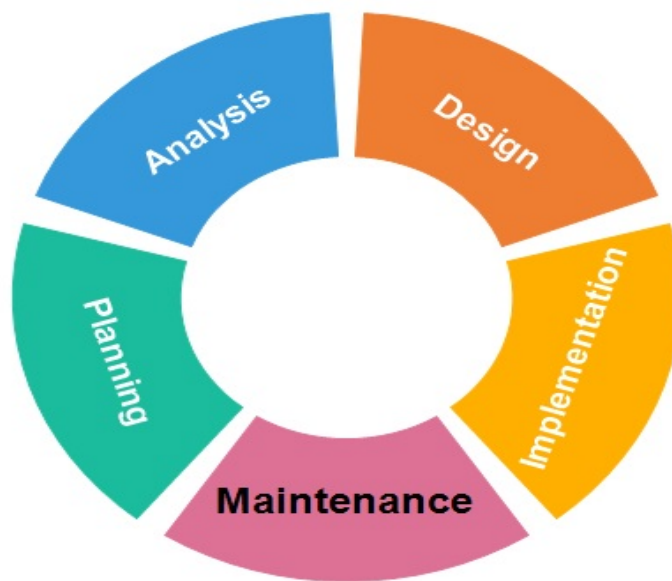


Figure 1.1: Software Development Life Cycle with the Maintenance Phase highlighted (Figure taken from 1)

Software Development Life Cycle¹ model is a framework describing well defined activities that are performed at each stage of software development project. The life cycle (refer Figure 1.1) begins from Project Planning where high level view of the

¹http://en.wikipedia.org/wiki/Systems_development_life_cycle

problem is established followed by Analysis phase where requirements, benefits, cost and alternative solutions of the problem are proposed and discussed. Then comes the system Design phase in which desired operations and features are defined elaborately. Following it is the Implementation Phase where real source coding is done. The continued process of improving the process performance is carried out in Maintenance phase (2). Software Maintenance¹ is the process of improving a product's quality after its delivery. It is required as failures keep on occurring and need of improvement keeps on growing throughout a product's cycle (3). It can be of various types¹- Corrective where faults are corrected in the hardware or software, Adaptive where software is made adaptable to a new environment, Perfective where performance is improved by implementing new features and requirements and Preventive, where problems are dealt before they occur by increasing the software reliability. Preventive maintenance is mostly done through refactoring in which software is changed in order to make it easier to comprehend and inexpensive to alter without making any changes in its observed external behavior (4). Issue Tracking Systems (ITS) which are software applications to update, maintain and resolve an issue, are a tool to guide the software maintenance process. Defect reporting and tracking which is critical in maintenance of software is made simple and efficient with the help of Issue Tracking Systems. Through an ITS, an issue is raised and described by a user, is classified, its status and progress is can be tracked until it gets resolved with the provision of commenting on error reports and feature requests. With the help of an ITS without wasting any time, useful information can be viewed directly like who is responsible for which issue, the current bug status, the priority with which bug it is to be resolved, preventing important issues from getting lost or delayed. Due to the usefulness of the services provided by an ITS, there are a lot of different ITS available in market for different projects. Bugzilla², Jira³, Mantis⁴ and Trac⁵ are the topmost ITS which are most popularly used now a days.

Popular ITS Bugzilla is a open source bug tracking system, which is used by Mozilla project for keeping the records of its unfinished bugs effectively. Figure 1.2a shows the main page of Bugzilla providing the important utilities to file a bug, search it, open a

¹<http://agile.csc.ncsu.edu/SEMaterials/MaintenanceRefactoring.pdf>

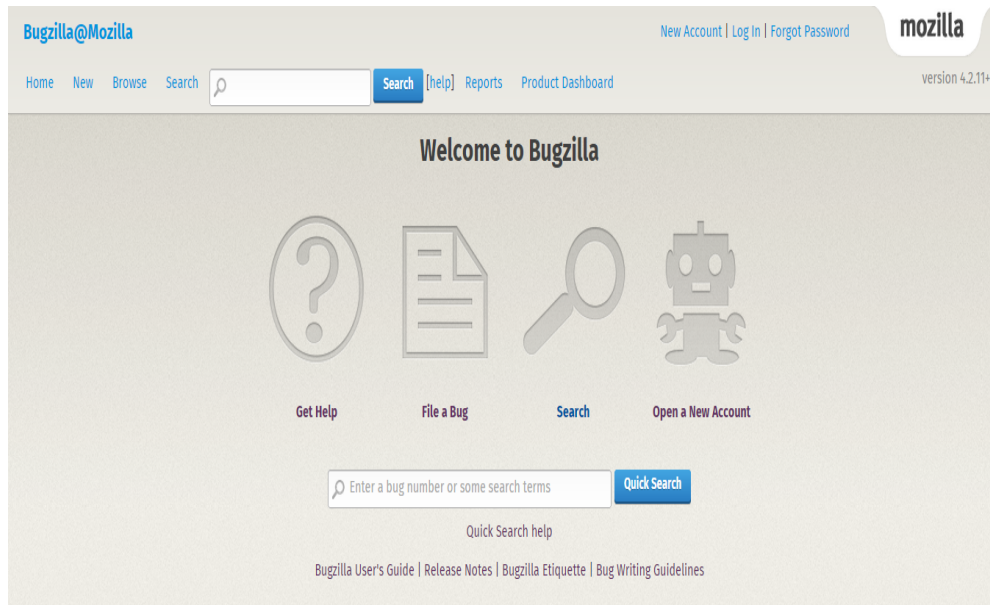
²<https://bugzilla.mozilla.org/>

³<https://www.atlassian.com/software/jira>

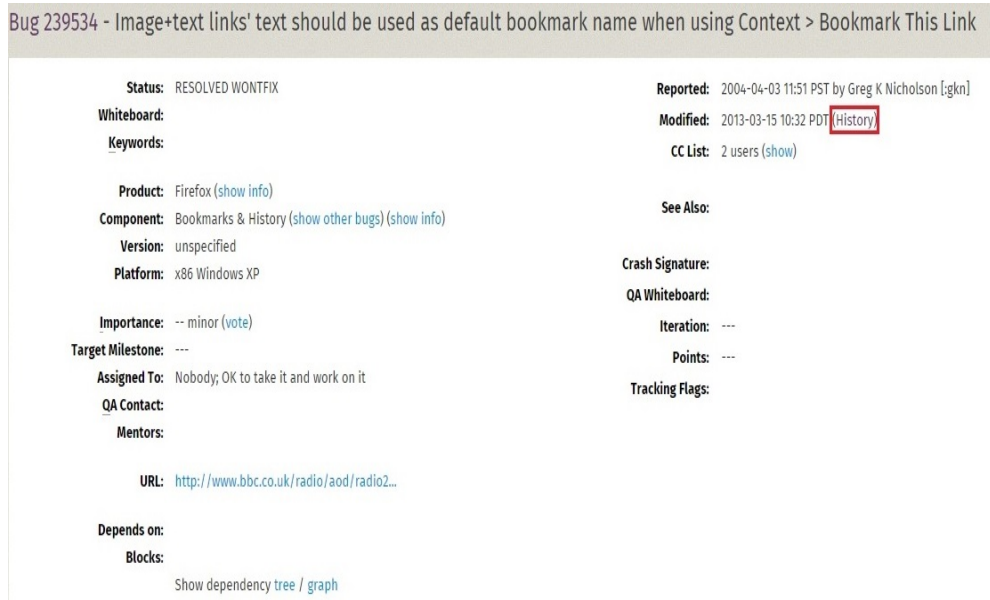
⁴<https://www.mantisbt.org/>

⁵<http://trac.edgewall.org/>

1.1 Issue Tracking Systems



(a) Bugzilla Main Page



(b) Bug details in Bugzilla

1.1 Issue Tracking Systems

Who	When	What	Removed	Added
mconnor	2004-04-03 16:33:00 PST	Status	UNCONFIRMED	NEW
		Ever confirmed	0	1
mconnor	2006-08-27 05:57:25 PDT	QA Contact	mconnor	bookmarks
mak77	2009-03-24 04:59:05	Assignee	p_ch	nobody
		CC		mak77
jaws	2013-03-15 10:32:42 PDT	Resolution		WONTFIX
		Last Resolved		2013-03-15 10:32:42
		Status	NEW	RESOLVED
		CC		jAWS

Figure 1.3: Snapshot of Mozilla Bug History (Bug ID 239534)

new user account and get associated help about any product of Mozilla or report any troubleshooting.

Figure 1.2b shows the attributes of a reported bug. Various fields displays the important information about the bug for instance, Status- its current resolution, Importance- its priority, Assigned to- its assignee etc. The History link available (highlighted with a red rectangle in Figure 1.2b), takes user to all the changes that occurred during the bug-fixing. Figure 1.3 shows the aggregated bug's history informing who created an event, when it happened and what actions were performed. This archived data can be mined to obtain useful results that can help in the improvement of the system.

In Bugzilla, a bug's life cycle consists of various well defined stages through which it goes as shown in Figure 1.4 ¹. It either enters the system as Unconfirmed where it requires confirmation of its existence or is already confirmed and enters as New state. From New state control can go to Assigned state where the bug is assigned to an appropriate person or the bug may directly be resolved and the status is set as Resolved. The path after Resolved state may lead to Verified state where the fix is

¹<https://www.bugzilla.org/docs/2.18/html/lifecycle.html>

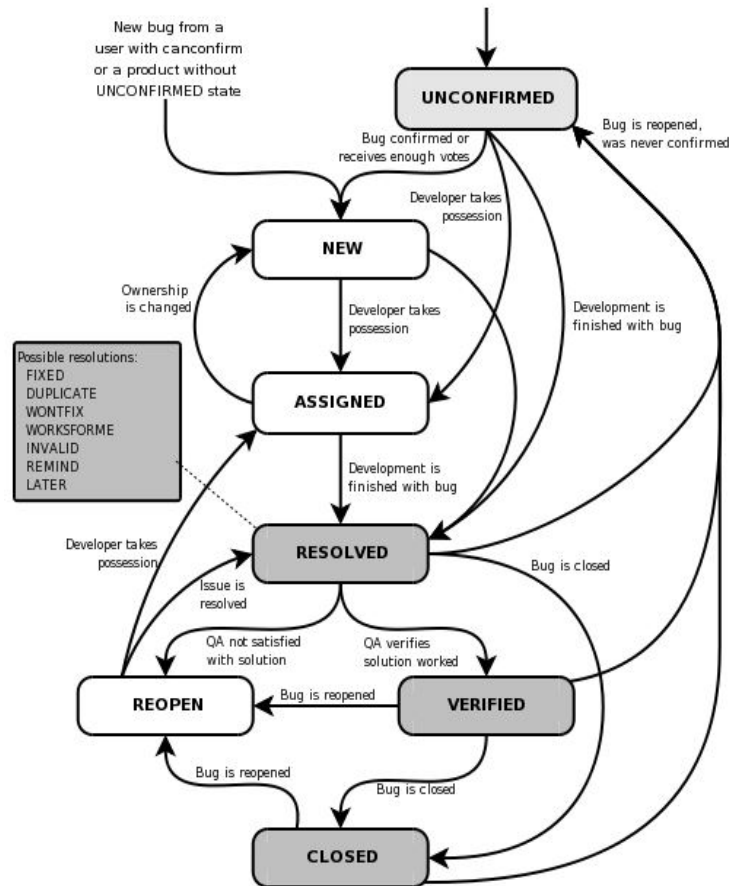


Figure 1.4: Bug Life Cycle in Bugzilla (Figure taken from 1)

expected to be validated by a Quality Assurance or if the verification is not performed bug may directly be Closed. A reopening may occur from Resolved state- if the QA is not satisfied with the solution, from a Verified or Closed state- if some additional information appears later. After reopening, assignments of proper person to the bug may happen if the resolution requires to be reviewed again or the resolution can be set directly.

1.2 Process Mining

Process mining is extraction of insights, consumable results and actionable information from event logs recorded by Process Aware Information Systems (PAIS). A PAIS is *a software system that manages and executes operational processes involving people,*

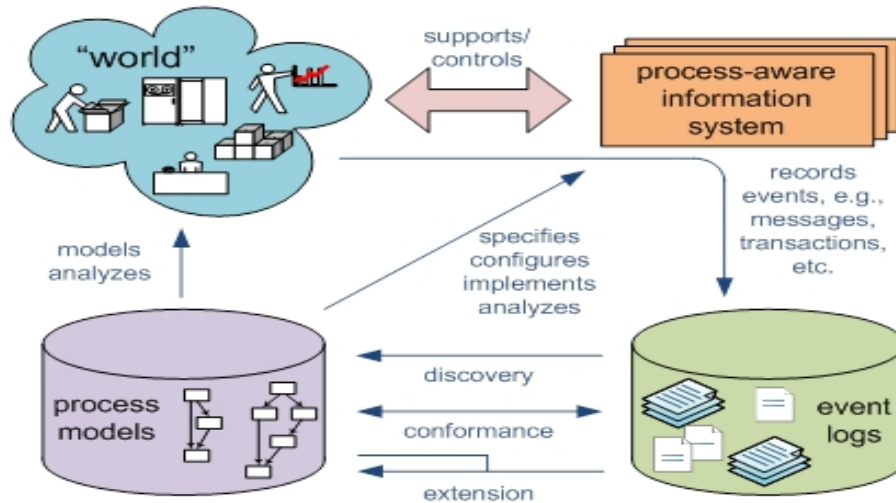


Figure 1.5: Process Mining Types (a) discovery (b) conformance and (d) enhancement (Figure taken from (1))

applications, and/or information sources on the basis of process models (5). Workflow Management systems and Business Process Management systems are examples of PAIS. An event log records all the developments of the process in execution. Event logs used for mining valuable insights consists of Case Id (process instance for which events are recorded), an activity (a distinct step in the process), an actor (an entity starting and executing the activity) and timestamps (the starting time of the event). All events belonging to a particular Case Id form a trace. From the data available in the event log, a process model can be created which helps in analysis of the entire system. A process model is a visual representation of the work flow of a business process that has been reverse engineered from an event log. It is a directed graph consisting of nodes and edges. Nodes represent the actions, known as activities that are performed in the business process. Edges denoted by arrows represent the transitions between these activities. If activity P occurs before activity Q in the given process, then there will be a directed edge from P to Q. Several notations have been proposed to represent a process model such as petri nets, casual nets, fuzzy models, UML activity diagrams etc.

Some of the added advantages¹ of Process Mining includes finding bottlenecks- as

¹<http://www.bptrends.com/the-added-value-of-process-mining/>

it equitably and automatically detects where the delays occur in the system, reducing waste- actual behavior is clearly evident in process models as the hidden additional activities which should not occur but are executing can be eliminated from the real system, ensuring compliance- as the deviations from the expected real life model are seen clearly in the process models, they can easily be measured, and promoting best practices- by comparing and analysing how the same processes are carried differently and selecting the best among them.

Process mining can be conducted to perform discovery, conformance or enhancement (6). Discovery technique takes an event log as input and produces a process model. Doing this helps in recovery of hidden and unknown facts that may help in the improvement of overall process. Conformance checking is used to check if the actual happenings of the system as recorded in the event log conforms to the process model and vice versa. It is basically keeping a check on the current workflow of the system by observing all the deviations that are observed in the process models. Enhancement aims at improving the existing process models using information available in the event logs as the process models generated may indicate the need for enhancing the current standard of workflow of the system (6).

Different perspectives of process mining available are shown in Figure 1.5, (7):

1. Process Perspective: Also known as control flow perspective, it focuses on ordering of tasks aiming at generating process models from the event log (7).
2. Organizational Perspective: It aims at determining the Structure of the organisation on basis of the people involved, their roles and their relations with each other (7).
3. Case Perspective: It focuses on properties of process instances. Analysis is done by looking at the element values of cases (7).

Process mining generates process models which carry visual and actionable insights of the raw data. There are many software tools available to mine the event logs e.g. Disco, Prom¹ etc. Disco automatically creates insightful process models providing a lot of advanced features to make things simpler to understand by a process analyst. The files that can be imported must be in CSV, XLS, MXML or XES format. Figure

¹<http://www.promtools.org/doku.php>

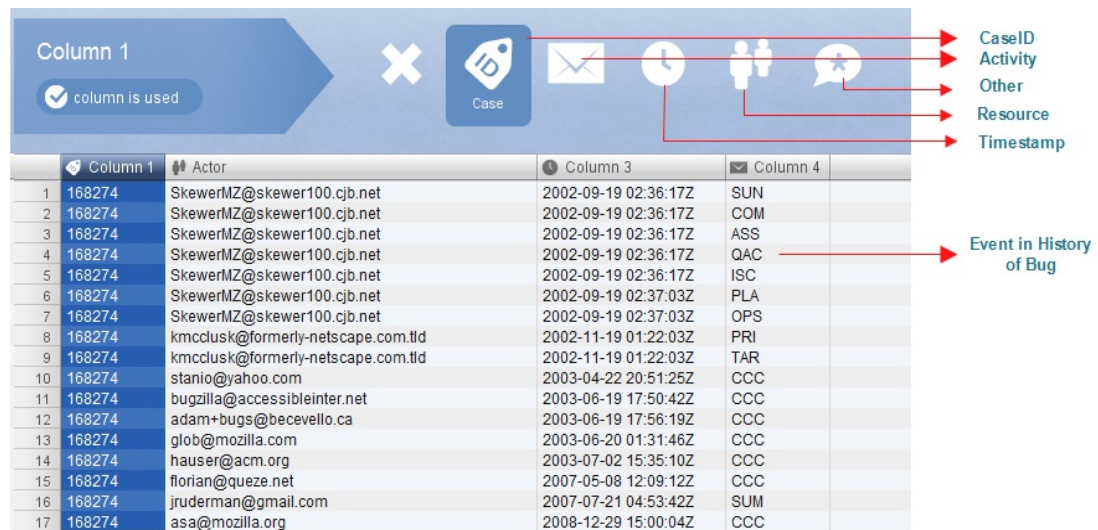


Figure 1.6: Snapshot of Disco after opening an Event Log in it. One can select the column and mark it as CaseID, Activity, Timestamp, Resource or Other.

1.6 shows an event log being imported in Disco to form a process model. Column 1 indicates the case id of each event, Column 2 the actor, Column 3 the timestamp and Column 4 the activity. Each column of the event log has to be selected and configured either as the Case Id, Activity, Actor (Resource), or the Timestamp. Atleast Case Id and Activity column has to be mentioned in the log without which the log would not be acceptable for the import. The pattern of timestamp entered has to be selected from the timestamps already available or it can be specified in Java's Simple Date format. In Disco, a zoom slider is provided that gives full control to the user to dig deeper into the model. Due to complex and confusing nature of real life event logs, Disco shows only the most important flows that is the most frequently occurring variants. Sliders for adjusting the number of activities and number of paths that are to be presented in the model are available. The process model has a starting (represented by a triangle symbol) and an end node (represented by a stop symbol) with all the activities, whose names are mentioned in the node itself, lying in between the two. Dashed arrows point to activities that occur at the very beginning or very end of the processes. Absolute Frequency of each transition is written over them. Nodes with more darker color and thicker edges signify a higher frequency count. Figure 1.7 shows a process model generated from an event log using Disco.

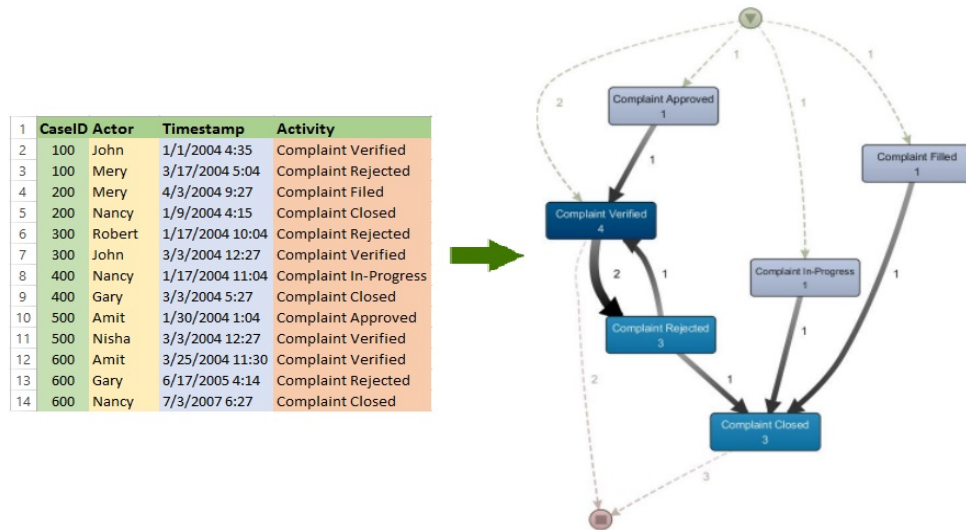
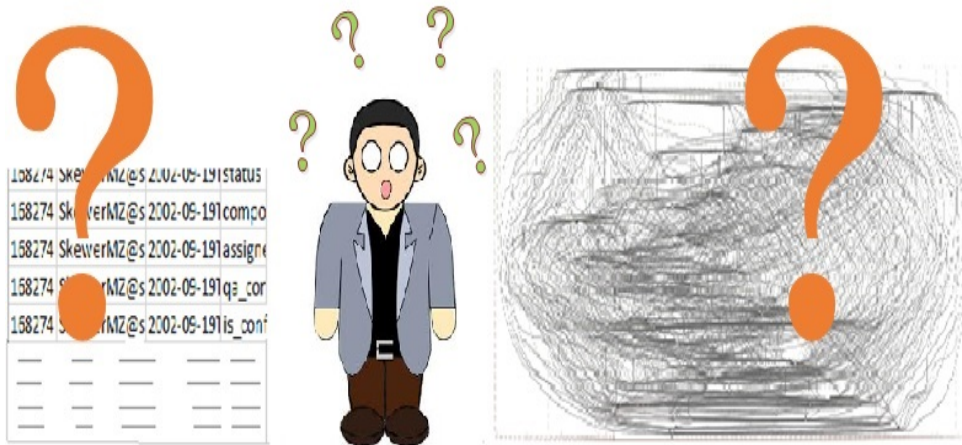


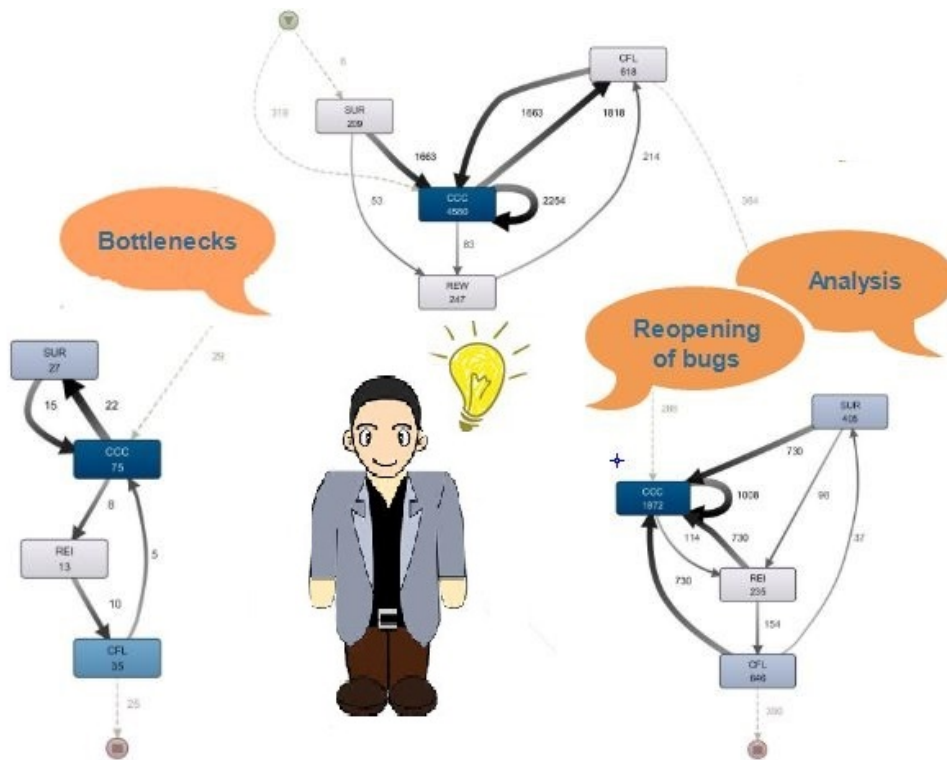
Figure 1.7: Process Discovery: Process Model generated from the given Event Log using Disco

For business process design and business process implementation, Business Process Modelling Notation-BPMN is used (8). BPMN process models comprises of two things- Activity nodes denoting activities performed by humans or software and Control nodes capturing the control flow between activities (9). Due to heterogeneity in the constructs of BPMN by the use of inconsistent terminologies in its definition, they are motivated to be converted into petri nets (9). Petri nets are a graphical as well as a mathematical notation for stepwise execution of a process . They are chosen to represent flow of either control, objects or information. They are composed of: Places and Transitions which makes the visualization easy to understand. Places contain tokens that flow in the system. Constraints that are enforced in petri nets are: having a distinguished source and sink and directing every transition between that unique source and sink only. There are several algorithms in literature, like alpha miner, flower miner, inductive miner etc. which can be used to create a petri net. The process model shown in Figure 1.7 represents a fuzzy model discovered from an event log using Disco showing the work-flow of a system.

Many real life event logs are unstructured, adhoc carrying a lot of diversity leading to the generation of 'spaghetti models'. These models are cumbersome to comprehend and only by zooming in one can get some level of understandability. For a process



(a) Process Analyst trying to analyse a Spaghetti Model



(b) Process Analyst studying the Process Models of Clusters

Figure 1.8: Problem of Spaghetti Model

analyst to understand either the unstructured big data logs or large spaghetti models is a challenge (refer Figure 1.8a). The solution to this can be dividing the spaghetti model into several simpler models by means of clustering (refer Figure 1.8b).

1.3 Problem Motivation, Definition and Aim

Software Process Intelligence (SPI) is an emerging and evolving discipline involving mining and analysis of software processes. This is modeled on the lines of application of Business Intelligence techniques to business processes (Business Process Intelligence (BPI)), but with the focus on software processes and its applicability to Software Engineering (SE) and Information Technology (IT) systems. Software Process Mining falls at the intersection of Software Process & Mining, and Software & Process Mining. SPI has diverse applications and is an area that has recently attracted several researcher's attention due to availability of vast data generated during software development. Some of the business applications of process mining software repositories or SPI are: uncovering runtime process models, discovering process inefficiencies and inconsistencies, observing project key indicators and computing correlation between product and process metrics, extracting general visual process patterns for effort estimation and analyzing problem resolution activities.

Several SE processes such as issue or defect resolution are flexible and consists of several process variants and a wide spectrum of behavior. This results in a spaghetti process model consisting of a large number of activity or task nodes as well as a large number of relations (or directed edges) between these nodes. A spaghetti process model is structurally complex and hard to comprehend for a process analyst. Trace clustering is a technique which has been applied on business process logs to split a given event-log into homogenous subsets from which process models are uncovered. Trace clustering has shown to improve the comprehensibility of process models in environments which allow process flexibility and large number of variants. The research motivation of the study presented in this paper is to investigate the application of trace clustering in the domain of SPI and process mining software repositories. The specific research aim of the work presented in this paper are the following:

1. To study the problem of spaghetti process models in the domain of software defect and issue resolution by conducting a case-study on open-source Firefox browser

1.3 Problem Motivation, Definition and Aim

project.

2. To propose a trace clustering technique based on grouping sequential data and apply it on issue tracking system dataset of a large, complex and log-lived open-source project. To investigate the effectiveness of the proposed trace clustering technique in reducing the structural complexity and enhancing the process model comprehensibility for a process analyst.
3. To study self-loops, back-and-forth, issue reopen, unique traces, event frequency, activity frequency and bottlenecks on the discovered process models from the homogeneous subset output of trace clustering and illustrate the benefits of trace clustering in the domain of SPI using a real-life case-study.

2

Related Work and Research Contributions

In this chapter we discuss previous work closely related to our study and list the novel research contributions of our work in context to already existing work.

2.1 Related Work

The related work has been categorised into three lines of research.

2.1.1 Problem of Spaghetti Process Models

Real life event logs are diverse, unstructured and complex leading to formation of 'Spaghetti models' which contain a lot of details without describing what is important and what is not (10). The problem of spaghetti process models has been discussed in (11). The paper discusses how mining becomes tougher with large unstructured data and also proposes the combined use of abstraction and clustering to make the process models easy to comprehend. Veiga et al. in (12) also examine the problem of complex spaghetti models and present an approach for representing only the essential information of these models by using sequential clustering in ProM. Authors in (10) suggests a mining approach that presents a simpler view of complex models so that they can provide useful abstractions of real-life processes.

2.1.2 Trace Clustering

Several techniques have been proposed in literature to cluster traces to deal with complex process models. Bose et al. propose a context aware approach to cluster process instances based on Levenshtein distance (13). In the technique substitution, insertion and deletion costs of symbols are derived for similarity. The authors evaluate the proposed algorithm on the telephone repair process event log and show that the approach is able to generate clusters with high degree of fitness and comprehensibility when compared to other approaches (13). In (11) Aalst et al. apply combination of abstraction and clustering techniques to simplify spaghetti-like models discovered using process mining techniques from unstructured and complicated processes (11). They use significance and correlation metrics to simplify the processes by clustering less significant but highly correlated data and removing less significant and correlated data from the simplified process model and implemented this technique as the Fuzzy Miner plugin for ProM (11). Ferreira et al. propose a sequence clustering approach where each cluster is represented by a first-order Markov chain. (14). The authors perform two different experiments to illustrate the effectiveness of the algorithm. In the first experiment recurrent interaction patterns among team members were discovered from event log data consisting of actions of a software development team while in the the second experiment common routines were discovered from the traces stored in a banking database using the proposed technique (14). Veiga et al. extended this work by using two dummy states (input and output state) with the Markov chain model for depicting the probability for an event to be the first or last in the sequence (12). They also suggest several preprocessing steps done before clustering to eliminate undesirable events from the event log (12). Weerdt et al. propose a new technique called ActiTraC (active trace clustering) for trace clustering which uses elements of active learning in an unsupervised environment (15). The proposed algorithm lessens the divergence between the clustering bias and the evaluation bias and improves the accuracy and complexity of process models (15). Song et al. In (16) propose a technique that cluster traces using several perspectives of traces such as performance, transition, case and event attributes organised as a feature vector. Each trace is represented by a trace profiles each describing different perspective. The authors use four different clustering techniques: K-means, Quality Threshold, Agglomerative Hierarchical Clustering and SelfOrganizing Maps using the

concept of trace profiles validate the approach by performing a real life case study (16). Greco et al. use a greedy trace clustering approach where process models are iteratively refined and each refinement leads to a more sound process model (17). The authors use a vector space model over the activities and their transitions for clustering traces present in the event log. The proposed technique cannot deal with loops and non-free-choice constructs (17). The trace clustering algorithm proposed by Medeiros et al. in (18) improves the technique proposed in (17) by overcoming several limitations. First a process model is generated from the entire event log using Heuristics Miner. If the model generated is optimal and does not suffer from over generalization stop otherwise cluster the event log using K-means. The clusters obtained are further partitioned if they are not optimal (18).

2.1.3 Conformance Measurement

Conformance is comparing the real behavior of the system with its expected behavior (19). Conformance measurement in business processes with the help of process mining has been shown in (19). The availability of event logs and control of users over some processes are informed to be the requirements for analyzing business alignment (19). The need of measuring conformance is not limited to only business sector, but is also required in performing security audits (20). Accorsi et al. have done a case study in financial sector that uses process mining techniques like conformance for auditing of security requirements (20). Rozinat et al. describe two metrics/dimensions of conformance testing namely, fitness and structural appropriateness (21). The authors show that both of these dimensions are needed to completely quantify conformance (21). They have implemented a conformance checker in ProM using which one can verify both of the metrics (21). In (22) authors propose two algorithms. One to measure fitness of a process model and another to find causes of inconsistencies between the process model and event log.

2.2 Novel Research Contributions

In context to existing work, the study presented in this thesis makes the following novel contributions:

2.2 Novel Research Contributions

1. Improving the goodness of process models by splitting the event-log into homogeneous subsets by clustering structurally similar traces by adapting the the k medoid algorithm.
2. Use of Longest Common Subsequence (LCS) or Dynamic Time Warping (DTW) distance metrics in the adaptation of K-medoid algorithm.
3. Illustrating the benefits of trace clustering in studying back-forth & self-loops, bug reopening and identifying bottlenecks.
4. An algorithm to automate clustering that returns the best cluster set for an event log by determining the goodness of process models.
5. An in-depth case study on the open source Firefox browser project to investigate the effectiveness of the proposed approach.

3

Research Framework and Solution Approach

Figure 3.1 shows the architecture diagram and the data processing pipeline for the Anvaya Framework. As shown in Figure 3.1, the data processing pipeline consists of 5 steps labeled as *A*, *B*, *C*, *D* and *E* respectively. Step *A* consists of extracting Issue Tracking System (ITS) data for the FireFox project using the Bugzilla REST API (an HTTP version of its XMLRPC and JSONRPC APIs)¹ and saving it in a MySQL Database. We extract the complete history (life-cycle) of all closed bugs [refer Algorithm 1]. The history consists of five fields: Who, When, What, Removed and Added. Each event in the history of a bug-report consists of a timestamp. An event in an event-log for a process model discovery algorithm requires a minimum of four fields: Case ID (or the Trace ID for the process instance), Actor, Timestamp and Activity. We map the ITS Issue ID as the Case ID, Who as Actor and a combination of What, Added and Removed as Activity.

We convert the history into an Event-log table consisting of three columns [Actor, Timestamp and Activity] where Activity column consists of the Activity-Id corresponding to What, Added and Removed in the Activity-Definition table. For this we extract, label and output all the unique activities from the Bugzilla history into an Activity-Definition table. For labelling, we use a three letter code which reflects and indicates the activities performed. Algorithm 2 shows the steps to create the Event Log and the Activity-Definition Table. We identify 81 unique activities in our data set. Table 3.1

¹ https://wiki.mozilla.org/Bugzilla:REST_API

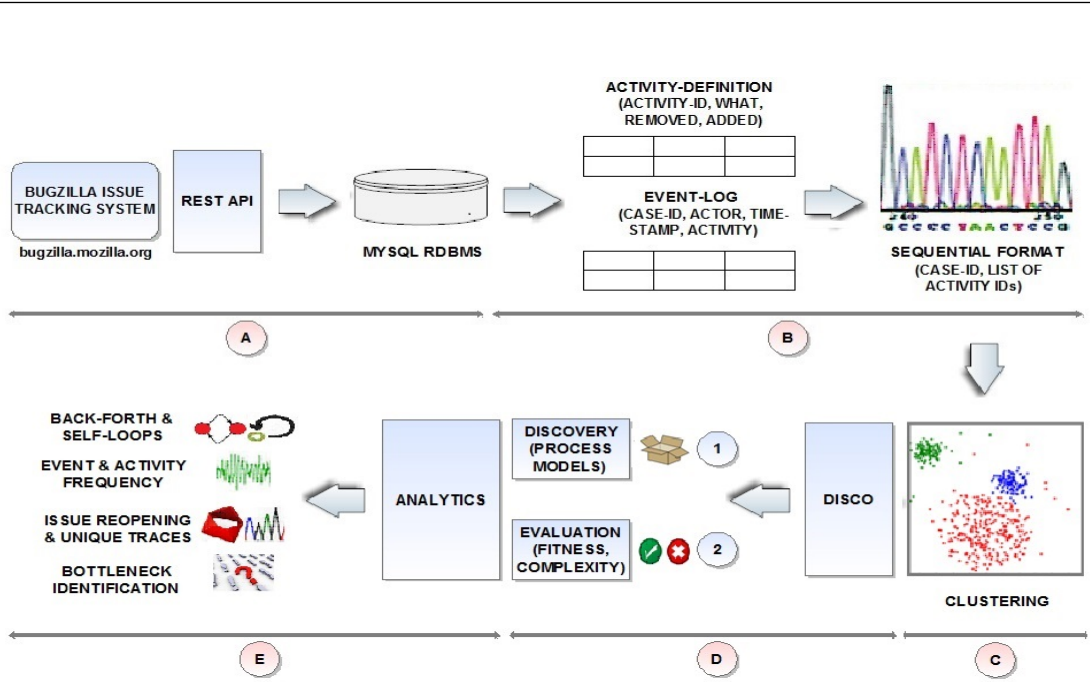


Figure 3.1: Architecture Diagram and Data Processing Pipeline for Anvaya Framework (Clustering-Based Approach for Improving the Goodness of Software Process Models Derived from Event-Logs)

shows the count and description of 20 unique Activity-IDs identified. We then transform this Event-log data into a sequential format [refer Algorithm 3] since clustering techniques can only be applied on sequential data. The data is in increasing order of Case IDs and activities within a case instance are in increasing order of timestamp. This step marks the end of Step *B* i.e. Data Transformation (refer Figure 3.2).

For Step *C*, we adapt¹ the K-medoid algorithm to cluster the sequential data using two different distance metrics: Longest Common Sub sequence and Dynamic Time Warping. Output of this step is a set of *k* clusters. The clustering algorithms are explained in detail in Section 5.

We now generate a single process model from the entire event-log data as well as for each cluster obtained in Step *C* using a commercially available tool Disco¹ that uses the fuzzy miner algorithm (10). We choose Disco because of its ability to manage large event logs and produce complex models. A node in the process model obtained from

¹Disco is a process mining toolkit for which we obtained the academic license.

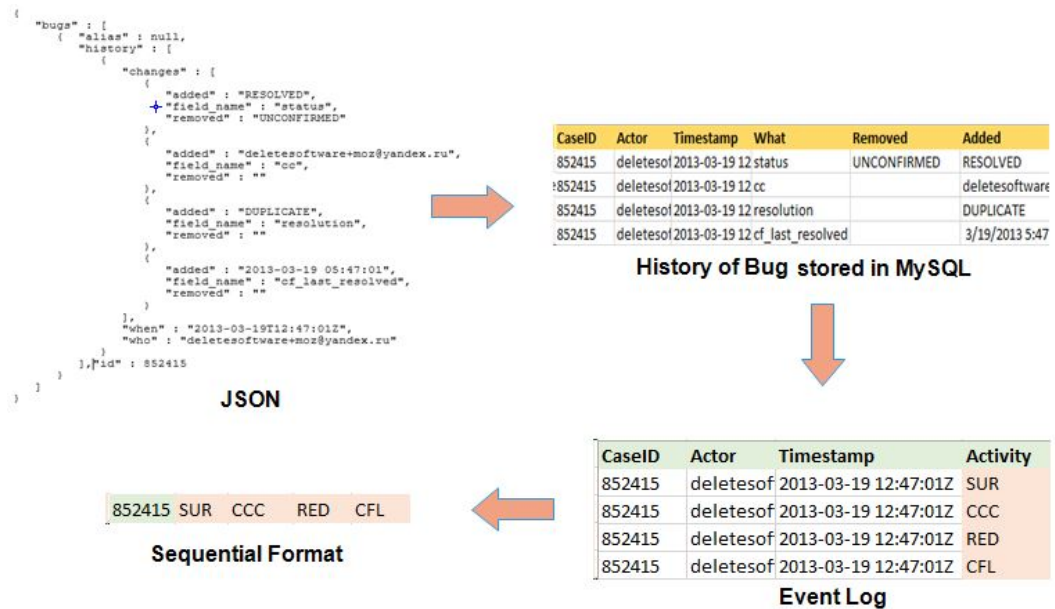


Figure 3.2: Data Transformation

Disco represents an Activity while an edge represents transition from one activity to another. We observe that the process model generated from the entire data is too complicated, spaghetti like and hard to comprehend where as process models generated by clustering structurally similar traces are simpler to understand and analyze. We evaluate the goodness of these process models using cyclomatic complexity and fitness metrics. The process models generated from the clusters are found to have high-degree of fitness and low degree of structural complexity.

The last step of Anvaya framework is the Analytics Step where we study and mine useful information from the process models generated from the clusters and show benefits of trace clustering in analysis of back-forth & self loops, bug reopening, and bottlenecks. Unique traces are discovered from the models and statistics for event & activity transitions are analyzed.

Algorithm 1: Data Extraction

Data: Rest API to access Bugzilla public data

Result: History data for all Firefox bugs closed in year 2013.

```
1 initialize database
2 foreach bugid  $b_i$  in Bugzilla do
3   | extract all data  $d_i$  for  $b_i$ 
4   | read  $d_i$ 
5   | if ( $d_i$  Status field equals "Verified" or "Resolved", Product field equals
6   | "Firefox" and Last_modified field starts with "2013") then
7   | | go to history section of  $b_i$ 
   | | download and write to database all events of  $b_i$ 
```

Algorithm 2: Event Log Creation

Data: History data for all Firefox bugs closed in year 2013.

Result: Event log of Firefox bugs closed in year 2013

```
1 foreach line  $l_i$  in history data do
2   | read and split  $l_i$  on each column
3   | generate activity id  $id_i$  from values of column fields
   | "what", "removed", "added"
4   | add  $id_i$  to act_ids []
5   | write values of column fields "bugid", "who", "when", and  $id_i$  to Event Log
   | file.
6   | if  $id_i \notin$  act_ids [] then
7   | | write  $id_i$ , and values of column fields "what", "removed", "added" to
   | | Activity-Definition Table
```

Algorithm 3: Sequential Data Creation

Data: Event log of Firefox bugs closed in year 2013.

Result: Event log in sequential data format.

```
1 set previous=null;
2 foreach line  $l_i$  in event log do
3   read and split  $l_i$  on each column.
4   if  $b_i \neq$  previous then
5     write  $b_i$  and  $id_i$  to sequential file
6     set previous=  $b_i$ 
7   else
8     append  $id_i$  to  $b_{i-1}$  in sequential data file
```

Table 3.1: Count and Description for some Activities.

Activity	Acronym	Count	Description
Alias	ALI	36	Short name assigned to bug for referring it at other places in Bugzilla.
Assigned to	ASS	4274	Bug is assigned to the proper person for setting its resolution.
Carbon Copy	CCC	48387	Users who are interested in the progress of the bug are included in the mailing list.
Component	COM	2765	Changing of the Component of the bug.
Depends On	DEP	6099	The bugs listed here are the ones on which this bug depends, so they must be resolved before this bug can be resolved.
Is Confirmed	ISC	1106	Confirming that the issue raised is a bug.
Platform	PLA	1312	Adding/removing the Platform of the bug.
Product	PRO	402	Changing the Product Category of the bug.
Resolution Fixed	REF	3876	A fix for the bug is determined and tested.
Resolution Invalid	REI	2879	The issue raised is not a valid bug and resolution is thus set to 'Invalid'.
Status Assigned Resolved	SAR	2344	The bug status changes from Assigned, where it was assigned to proper person for setting its resolution to Resolved where resolution has been performed and is awaiting verification by Quality Assurance.
Status Assigned Unconfirmed	SAU	3	The bug status changes from Assigned, where it was assigned to proper person for setting its resolution to Unconfirmed where it is validated whether the bug is true or not.
Status New Resolved	SNR	4492	The bug status changes from New, where it was processed and resolved to Resolved where it is awaiting verification by Quality Assurance.
Status Reopened New	SRN	41	The bug status changes from Reopened where the bug was reopened as the resolution was later found to be incorrect to New where it is assigned for processing.
Status Resolved Reopened	SRR	702	The bug status changes from Resolved, where its resolution was set, to Reopened where the bug is reopened as the resolution is found to be incorrect.
Status Resolved Verified	SRV	731	The bug status changes from Resolved where resolution has been performed to Verified where Quality Assurance has looked at the bug and its resolution and agrees that the appropriate resolution has been performed.
Status Unconfirmed Assigned	SUA	76	The bug status changes from Unconfirmed where it is validated whether the bug is true to Assigned where it is assigned to the proper person for processing.
Summary	SUM	2362	The short sentences describing what the bug is about are added/removed.
Status Unconfirmed Resolved	SUR	5334	The bug status changes from Unconfirmed where it is Validated whether the bug is true to Resolved where resolution has been performed and it is awaiting verification by Quality Assurance.
Version	VER	1663	Changing of the version of the software in which the bug was found.

4

Experimental Dataset

Table 4.1: Experimental Dataset Details (Mozilla Firefox Project)

Attribute	Value
Project	Firefox
First Issue Report Date	1 January 2013
Last Issue Report Date	31 December 2013
Data Extraction Date	16 October 2014
Number of Open Issues	3399
Number of Closed Issues Used	11804
Number of Activities in Closed Issues	81
Number of Events Reported for Closed Bugs	178331

We conducted a case-study on one of the largest open-source bug tracking system Bugzilla. We use closed bug report data for Firefox Browser because closed bugs have completed their lifecycle. We do not analyse open bug report data because such bugs are still in the pipeline, work is being done on them, and we don't know what shape they are going to take. We Extract bug report for data Firefox Browser having reporting timestamp of closure from 1st January 2013 to 31st December 2013 (12 months data). Table 4.1 shows the experimental dataset details for the Mozilla Firefox project. We conduct experiments on publicly available dataset so that our approach or results can be replicated and used for benchmarking and comparison. We share our dataset and

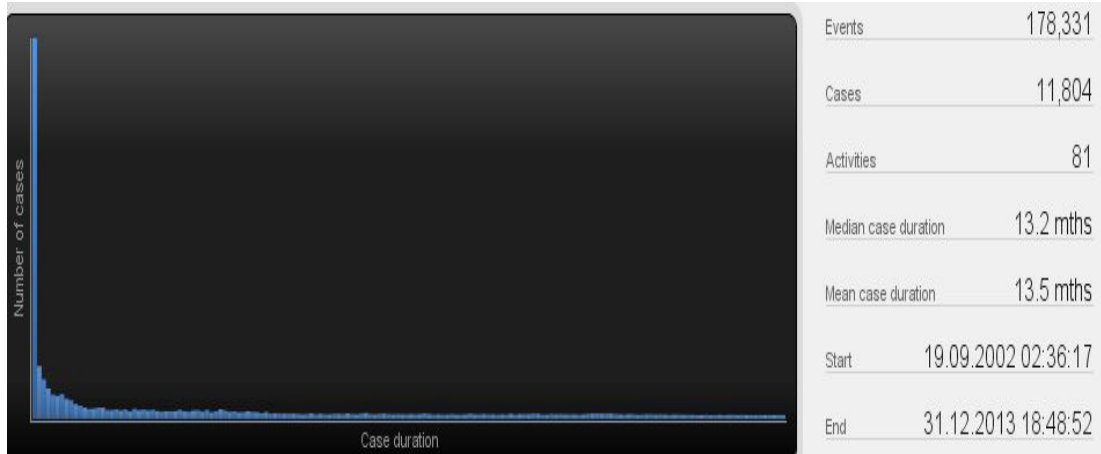


Figure 4.1: Case Duration

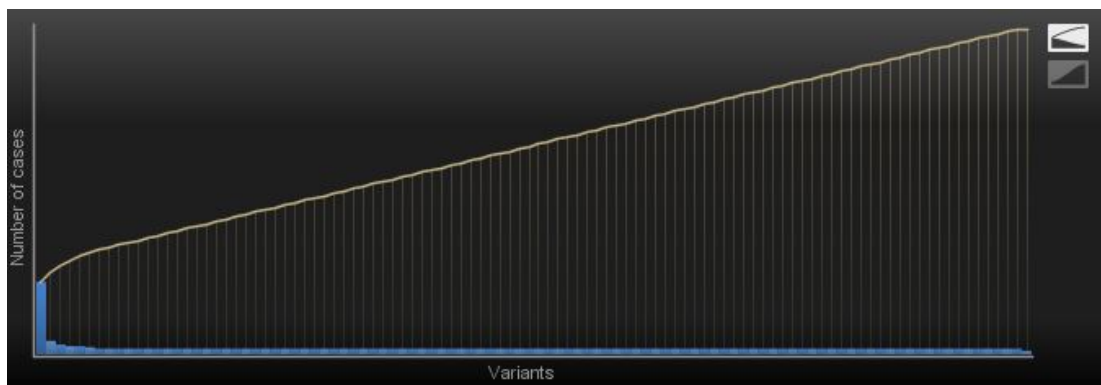



Figure 4.2: Number of Cases vs Case Variants


associated files by creating a public repository on GitHub¹.


There are 11804 closed issues/process instances and 178331 records/events in the Event Log. We have identified 81 unique activities from this dataset. The bug history dataset consists of five fields, namely Who, When, What, Removed and Added. Who specifies the name of the Actor who performs an Activity at a particular time. When field specifies the Timestamp when the particular activity was performed. Combination of What, Removed and Added has been taken as the activity. In What field bug fields are specified whose value can be added or removed. Some of the examples of bug fields are Status, Component, Content, Comment Tag, Resolution, Keywords, OS, Priority

¹<https://github.com/ashishsureka/anvaya>

CaseID	Who	When	What	Removed	Added
168274	SkewerMZ@skewer100.cjb.net	2002-09-19 02:36:17Z	status	UNCONFIRMED	NEW
168274	SkewerMZ@skewer100.cjb.net	2002-09-19 02:36:17Z	component	Browser-General	HTML Form Controls
168274	SkewerMZ@skewer100.cjb.net	2002-09-19 02:36:17Z	assigned_to	asa@mozilla.org	jkeiser@netscape.com
168274	SkewerMZ@skewer100.cjb.net	2002-09-19 02:36:17Z	qa_contact	asa@mozilla.org	tpreston@netscape.com
168274	SkewerMZ@skewer100.cjb.net	2002-09-19 02:36:17Z	is_confirmed		1
168274	SkewerMZ@skewer100.cjb.net	2002-09-19 02:37:03Z	platform	PC	All
168274	SkewerMZ@skewer100.cjb.net	2002-09-19 02:37:03Z	op_sys	Linux	All
168274	kmclusk@formerly-netscape.com.tld	2002-11-19 01:22:03Z	priority	--	P4
168274	kmclusk@formerly-netscape.com.tld	2002-11-19 01:22:03Z	target_milestone	---	Future
168274	stanio@yahoo.com	2003-04-22 20:51:25Z	cc		stanio@yahoo.com
168274	bugzilla@accessibleinter.net	2003-06-19 17:50:42Z	cc		based@free.fr
168274	adam+bugs@becevello.ca	2003-06-19 17:56:19Z	cc		abecevello@sympatico.ca
168274	glob@mozilla.com	2003-06-20 01:31:46Z	cc		bugzilla@glob.com.au


CaseID


Actor


Timestamp



Activity

Figure 4.3: Mozilla Firefox Event Log Data

etc ¹. Figure 4.3 shows the Event Log of the Experimental Dataset used.

We extract and store the dataset in MySQL database from where it can be extracted as a CSV (Comma Separated Values) file format which can be opened in Disco where it is mandatory to select columns as CaseID and Activity. We then select and configure the columns that contain CaseID, Timestamp and Activity and obtain Figures 4.1 and 4.2 from the Statistics View of the Disco tool. Statistics view provides the detailed statistics information and performance metrics of our event log ². Figure 4.1 shows the graph between Number of Cases and Case Duration. As seen from the figure, most of the cases have short duration and there are extremely few cases with large duration. The median case duration is 13.2 months while the mean case duration is 13.5 months. Figure 4.2 shows the graph between Number of Cases and case Variants. Variants are the unique traces present in the Event log. There are 8697 variants present in our Experimental Dataset. Large number of variants show that bug life-cycle consists of diversity and variance.

¹<https://bugzilla.mozilla.org/page.cgi?id=fields.html>

²fluxicon.com/disco/files/Disco-User-Guide.pdf

5

Clustering

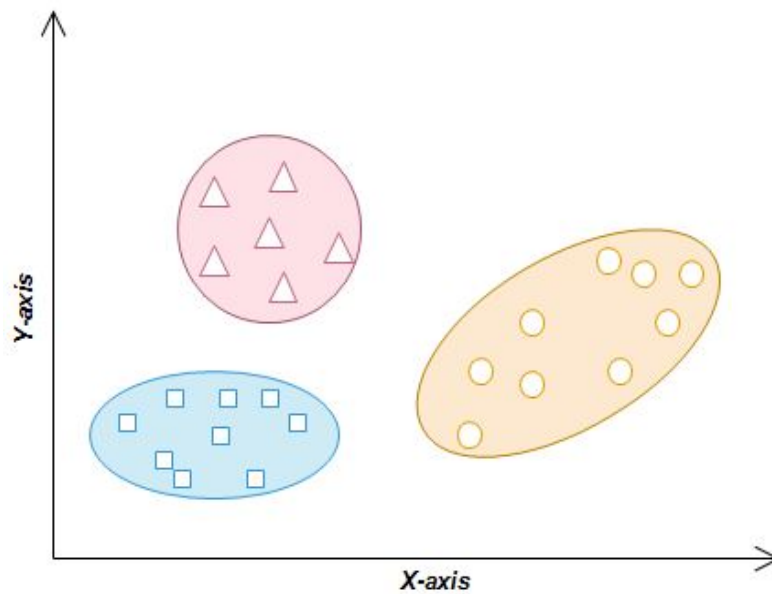


Figure 5.1: Clustering

Clustering is grouping similar objects together. An object in a cluster is similar to the rest of the objects of its group but dissimilar to objects belonging to other groups. As can be seen from Figure 5.1, objects having similar shape are clustered together. Shape of objects in a cluster are different from shape of objects in other clusters. Clustering is an unsupervised learning technique as unlike supervised learning, target class values or labels are not available a priori. It can be of

many types.¹ *Connectivity based clustering* (also known Hierarchical clustering) builds a hierarchy of clusters. This clustering joins objects to form a cluster based on the distance between them. It can be of two types: agglomerative (bottom-up approach) and divisive (top-down approach). In *centroid based clustering*, a cluster is represented by a central vector which may or may not be an element of the dataset. The algorithm finds k centers and assigns each object to the nearest center. *Distribution based clustering* is related to statistics and cluster objects belonging to same distribution. In *Density based clustering* areas with higher density are defined to be clusters and areas of sparser densities are taken to be noise. We have adapted the k medoid algorithm, a medoid based clustering approach to cluster the sequential traces.

5.1 K Medoid Clustering

K medoid clustering algorithm is a partitional clustering approach where each cluster is represented by a medoid which is the most centrally located data point in a cluster whose average similarity to all other data points in that cluster is maximal (23). Medoids differ from centroids as they are always members of the given dataset² making this algorithm insensitive to outliers. This algorithm partitions the datasets into k clusters such that distance between the data points assigned in a cluster and center of that cluster is minimized³. Determining the number of clusters (k) is out of scope of this research work and we plan to study and implement it in future. We are assuming that the software quality assurance manager, process analyst or any other user of Anvaya Framework is qualified enough to have good insights about the data. He has the intuition and hunch about the number of clusters that can be created from the dataset. As seen from Figure 5.2, the end user of Anvaya Framework uses his domain knowledge as well as gut feeling to determine k in an iterative process.

Algorithm 4 describes the steps to compute k clusters using our proposed technique. Initially k traces are selected as initial medoids. For initial cluster assignment, we compute the distance of each non medoid trace with all medoid traces. We propose the use of two popular algorithms: Longest Common Subsequence (LCS Similarity) and

¹http://en.wikipedia.org/wiki/Cluster_analysis

²<http://en.wikipedia.org/wiki/Medoid>

³<http://en.wikipedia.org/wiki/K-medoids>

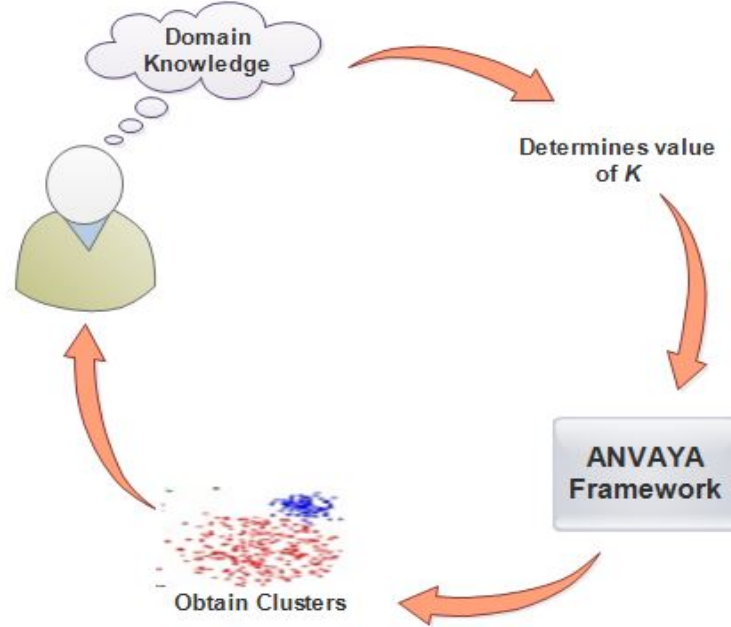


Figure 5.2: Software Quality Assurance Manager using his Domain Knowledge to Determine the value of K in an Iterative Process

Dynamic Time Warping (DTW Similarity) as the distance metrics which calculate the similarity score (lcs_i or dtw_i) between two traces. The former metric returns the length of longest common subsequence while the latter returns the warping distance (higher the warping distance, lesser is the similarity). So a non medoid trace is associated to a medoid with highest lcs_i or lowest dtw_i . To select the best configuration each non medoid trace is swapped with each medoid trace and total similarity score (cost) is calculated. The configuration with the highest cost while using LCS Similarity and lowest cost while using DTW Similarity is selected. The steps are repeated till there is no change in the medoids.

5.1.1 Longest Common Subsequence metric

The first distance metric that can be used to compute the similarity between two traces is the Longest Common Subsequence metric. Since each trace is nothing but a sequence of characters, we use the popular LCS algorithm to determine the length of the longest common sequence of characters which need not be consecutive but follow a left to right ordering as illustrated in Figure 5.3. LCS is a classic and popular problem which has

Algorithm 4: k Medoid Clustering

Data: Event log in sequential data format

Result: k clusters

- 1 input the value of number of clusters to be formed k .
 - 2 read the input event log
 - 3 randomly select k traces as initial medoids.
 - 4 **foreach** non medoid trace t_i **do**
 - 5 **foreach** medoid trace m_i **do**
 - 6 calculate similarity score of t_i and m_i using LCS Similarity lcs_i or DTW Similarity dtw_i
 - 7 assign t_i to m_i with highest lcs_i or lowest dtw_i .
 - 8 **foreach** medoid trace m **do**
 - 9 **foreach** non medoid trace o **do**
 - 10 swap m and o
 - 11 compute the total similarity score (cost) of the configuration using either lcs_i or dtw_i
 - 12 select the configuration with the highest cost while using LCS Similarity and lowest cost while using DTW Similarity.
 - 13 Steps 4 to 12 are repeated till there is no change in the medoids
-

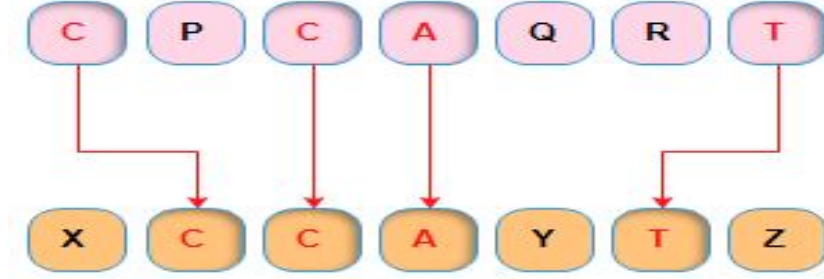


Figure 5.3: Longest Common Subsequence

been explored a lot in literature. Let $\text{Similarity}[i,j]$ be the length of LCS of sequences $S1_i$ and $S2_j$. The algorithm uses the following recursive formula: (24):-

$$\text{Similarity}[i,j] = \begin{cases} \text{Similarity}[i-1, j-1] + 1 & \text{if } S1[i] = S2[j] \\ \max\{\text{Similarity}[i-1, j], \text{Similarity}[i, j-1]\} & \text{otherwise} \end{cases}$$

Algorithm 5 gives the pseudocode of dynamic programming implementation of LCS problem (24). The algorithm takes two traces as input and returns their similarity score.

Algorithm 5: LCS Similarity

Data: Trace $s1$ and $s2$ from event log in sequential data format.

Result: Similarity score between the two input traces.

```

1 find the length  $n$  of  $s1$  .
2 find the length  $m$  of  $s2$  .
3 for  $i \leftarrow 0$  to  $n$  do
4    $\lfloor$  Similarity[ $i$ ][0]=0
5 for  $j \leftarrow 0$  to  $m$  do
6    $\lfloor$  Similarity[0][ $j$ ]=0
7 for  $i \leftarrow 1$  to  $n$  do
8   for  $j \leftarrow 1$  to  $m$  do
9     if  $s1[i-1]$  equals  $s2[j-1]$  then
10     $\lfloor$  Similarity[ $i$ ][ $j$ ]=Similarity[ $i-1$ ][ $j-1$ ] +1
11    else
12     $\lfloor$  Similarity[ $i$ ][ $j$ ]= max(Similarity[ $i-1$ ][ $j$ ], Similarity[ $i$ ][ $j-1$ ])
13 return Similarity[ $n$ ][ $m$ ]

```

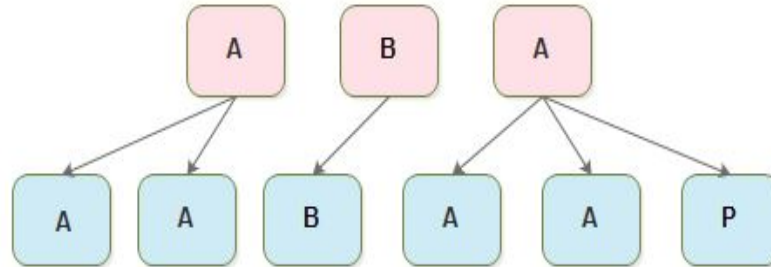


Figure 5.4: Dynamic Time Warping

5.1.2 Dynamic Time Warping metric

Dynamic Time Warping is a popular algorithm used in speech recognition, text mining, video retrieval and many other applications to find similarity between temporal sequences, having same or different lengths, that are non-linearly warped in time dimension i.e. they are structurally similar but are on a different timescale^{1 2}.

Let two sequences be $S1$ and $S2$. Warping path consists of index pairs (i,j) if DTW associates $S1[i]$ with $S2[j]$. It has certain restrictions, namely Monotonicity that says that the warping path cannot go backwards in time, Continuity that says that every character of each sequence is included in the warping path and the indices can increase by 0 or 1 and Boundary condition which states that the warping path begins at $(1,1)$ and ends at (n,m) where n and m is the length of the first and second sequence respectively (25). Out of the many warping paths, an optimal warping path is the one that minimizes the total cost (25). Warping distance is the summation of element wise distance between $S1[i]$ and $S2[j]$ over all pairs of (i,j) in the optimal warping path¹. We assign a cost (distance) 0 if $S1[i]=S2[j]$, otherwise 1 is assigned. Let $S1=ABA$ and $S2=AABAAP$, then the optimal warping path consists of indices $(1,1)$ $(1,2)$ $(2,3)$ $(3,4)$ $(3,5)$ $(3,6)$ and warping distance will be 1.

Algorithm 6 gives the pseudocode of the dynamic programming implementation of DTW algorithm. The algorithm returns the similarity score which is the warping distance. Lower the warping distance, more similar are the traces.

¹<http://cs.bc.edu/~alvarez/Algorithms/Notes/dtw.html>

²http://en.wikipedia.org/wiki/Dynamic_time_warping

Algorithm 6: DTW Similarity

Data: Sequences $s1$ and $s2$ from event log in sequential data format.

Result: Similarity value between the two input sequences

```
1 find the length  $n$  of  $s1$  .
2 find the length  $m$  of  $s2$  .
3  $cost=0$  .
4 for  $i \leftarrow 1$  to  $n$  do
5    $\lfloor$   $DTW[i][0]=\infty$ 
6 for  $j \leftarrow 1$  to  $m$  do
7    $\lfloor$   $DTW[0][j]=\infty$ 
8  $DTW[0][0]=0$ 
9 for  $i \leftarrow 1$  to  $n$  do
10  for  $j \leftarrow 1$  to  $m$  do
11    if  $s1[i-1]$  equals  $s2[j-1]$  then
12       $\lfloor$   $cost=0$ 
13    else
14       $\lfloor$   $cost=1$ 
15     $\lfloor$   $DTW[i][j]=cost+\text{minimum of}(DTW[i-1][j],DTW[i][j-1],DTW[i-1][j-1])$ 
16 return  $DTW[n][m]$ 
```

6

Evaluation

We evaluate two aspects of the process models, namely complexity and fitness. Process models generated from the clusters should exhibit low degree of structural complexity and high-degree of fitness.

6.1 Complexity

Complexity can have unwanted effects on understandability, comprehensibility and correctness of process models (26). Many complexity metrics have been proposed in literature (27) (28) which tells whether the process model is compact, has appropriate size, is easy to comprehend and understand. We use McCabe's cyclomatic number which determines the number of linearly independent paths in the process model (29). It represents complexity in a single number and represents the magnitude of all possible independent paths that can be followed in the model. The pseudocode to determine the cyclomatic number of process models obtained from Disco is given in Algorithm 7. The Xml format input of the process model is needed as it carries all the relevant information namely, the number of edges, nodes along with the adjacency matrix which is required for calculating the complexity. The higher the complexity value returned by this algorithm, higher will be number of independent paths and thus more complex will be the model.

Algorithm 7: Complexity

Data: Xml format input of the process model**Result:** Complexity of the process model

- 1 read number of edges e
 - 2 read number of nodes n
 - 3 complexity= $e-n+2$
-

6.2 Fitness

One of the major applications of Process Mining is to determine the gaps between the real world as recorded in the event log and the model¹. It helps in detecting whether there is problem with the real world execution of the system or if the existing model requires any updates. The Fitness metric is used to determine the conformance between an event log and a process model generated from that log. In a completely fit model each and every trace present in the event log is replicated in the process model. High fitness indicates that the process model majorly captures the behaviour of all traces in the event log. The pseudocode to determine the fitness of the process model at event level is given in Algorithm 8 (22). The fitness value of a process model can take any values between 0 and 1. Fitness value 1 indicates that the process model has a perfect one to one mapping with the event log while value 0 indicates that none of the traces present in the even log are shown in the process model.

¹http://www.processmining.org/online/conformance_checker

Algorithm 8: Fitness

Data: Xml format input of the process model and Event log in sequential format.

Result: Fitness of the process model.

```

1 read Xml format input file.
2 foreach transition between a source  $n_i$  and target node  $n_j$  do
3   | adjacency matrix  $a_{n_i, n_j} = 1$ 
4 read the input event log
5 foreach bug id  $b_i$  do
6   | add each activity to trace  $t_i$ 
7   | if  $t_i$  is unique then
8     | | add it to uiguetrace[]
9     | | Count its frequency  $F_i$  in the event log
10 foreach entry  $t_i$  in uiguetraces[] do
11   |  $Valid_i = 1$ 
12   |  $j = 1$ 
13   | while  $j < \text{length of } t_i$  do
14     | | if  $a_{t_i[j], t_i[j+1]} \neq 1$  then
15       | | |  $Valid_i = 0$ 
16       | | | break
17     | | else
18       | | |  $j++$ 
19 foreach entry  $t_i$  in uiguetraces[] do
20   |  $\text{FreqValidProduct} = \text{FreqValidProduct} + F_i * Valid_i$ 
21   |  $\text{FreqSum} = \text{FreqSum} + F_i$ 
22  $\text{Fitness} = \text{FreqValidProduct} / \text{FreqSum}$ 

```

7

Experimental Results

To validate the clustering, k medoid algorithm using LCS and DTW similarity metrics was applied on 1615 process-instances and 6 clusters were obtained.

Fig 7.1 shows the process models generated using Disco at 100% activity and 12.2% path resolution from the entire event log (referred as the main model throughout the paper) and event log of the six clusters obtained after applying k medoid with LCS distance metric. Table 7.2 contains the complexity and fitness values of main model and all the six clusters. The complexity has been reduced by 40% on an average in a cluster clearly showing that clusters are now much easier to comprehend and analyse. Process models of 66% clusters have a better one to one mapping with the event log and thus show a better fitness value. Table 7.1 characterizes the domain of the six clusters obtained using LCS distance metric.

Fig 7.2 shows the process models generated using Disco at 100% activity and 12.2% path resolution from the entire event log and event log of the six clusters obtained after applying k medoid with DTW distance metric. The complexity and fitness of main model and all the six clusters is shown in Table 7.3. The complexity has been reduced by 40.98% on an average in a cluster while process models of 83.33% clusters have better fitness values as compared to the main model.

Throughout our work in further sections, we have used LCS distance metric for analysis.

Table 7.1: Cluster Description

	Description
Cluster 1	100 bugs are reported in all OS, 42 in Windows XP and 33 in Mac OS and Windows 7. Most of them are reported in platform x86 with severity field Normal and current resolution Worksforme or Wontfix. The status of bugs is first modified from Unconfirmed to New (SUN) and it is then confirmed to be true (ISC). For most of the bugs the status then changes from New to Resolved where it is awaiting verification by Quality Assurance (SNR) while for some the status changes directly from Unconfirmed to Resolved (SUR). Resolution of the bugs is then set to either Wontfix (REX) or Worksforme (REW) before the last resolved stage (CFL). Bugs are reopened after the resolution is set as Fixed.
Cluster 2	94 of the bugs are reported in all OS, 52 in Windows 7 and 46 in Windows XP. Most of them are reported in platform x86 with severity field Normal and current resolution Worksforme or Wontfix. In most of the bugs status is first modified from Unconfirmed to Resolved after which their resolution either changes to Wontfix (REX) or Worksforme (REW) before the last resolved stage (CFL). While for some bugs, version of the software the bug was found in (VER) is added, various flags (FLA) are set, tags and status information is added in text entry box of bug (WHI), and blocks (BLO) field is added, status is changed from New to Resolved (SNR) after which the resolution either becomes Wontfix or Worksforme.
Cluster 3	Almost all bugs are independent of OS and platform. Most of them are reported with severity field Normal and current resolution Fixed. The bugs are assigned to the proper person (ASS). Summary of bugs (SUM), keywords (KEY), flags (mostly attachment flags), blocks (BLO) field and Target Milestone (TAR) field is added. Properties of attachments (ATT) are added/removed. Bugs are identified and listed in Depends on field (DEP). Bugs listed here must be resolved before the bug can be resolved. Status changes from New to Resolved. For most of the bugs resolution becomes Fixed (REF) and for others it becomes either Wontfix (REX) or Worksforme (REW) before the last resolved stage (CFL).

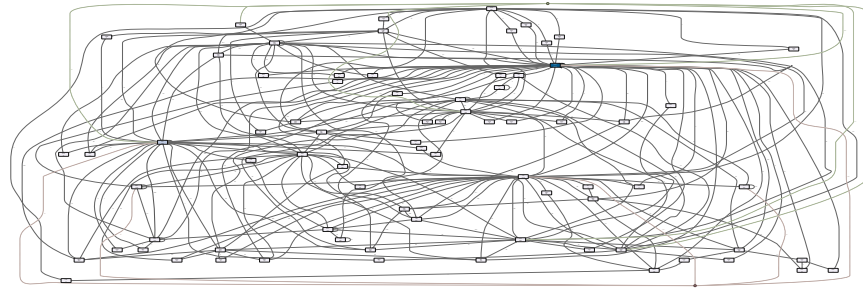
Cluster 4	121 bugs are reported in all OS, 68 in Windows XP and 37 in Linux. Most of them are reported in all or x86 platform with severity field Normal or Enhancement. Current resolution of 101 bugs is Wontfix and 102 bugs is Worksforme. Status of bugs changes from New to Assigned (SNA) and it is then confirmed to be true (ISC). Component (COM) field belonging to the product and person responsible for confirming if bug is unconfirmed, and for verifying the fix is added and the bugs are assigned to the proper person (ASS). For most of the bugs the status then changes from New to Resolved (SNR) while for some the status changes directly from Unconfirmed to Resolved (SUR). Resolution of the bugs is then set to either Wontfix (REX) or Worksforme (REW) before the last resolved stage (CFL).
Cluster 5	161 bugs are reported in all OS, 38 in Windows 7 and 25 in Mac OS and Windows XP. Most of them are reported in all platforms with severity field Normal and current resolution Fixed. The bugs are assigned to the proper person (ASS). Several flags (FLA), keywords (KEY) to easily identify and group the bugs are added and tags & status information is added in text entry box of bug (WHI). Properties of attachments (ATT) are added/removed. Bugs are identified and listed in Depends on field (DEP). Status either changes from Assigned to Resoved (SAR) or New to Resolved (SNR), Target Milestone (TAR) field is defined and Resolution becomes fixed (REF) before the last resolved stage (CFL).
Cluster 6	75 bugs are reported in Windows 7 and 54 in Linux . Most of them are reported with platform x86 and severity field Normal or Major. Current resolution of 108 bugs is Incomplete and 73 bugs is Worksforme. Version of the software the bug was found in (VER) is added, various flags are set and tags & status information is added in text entry box of bug (WHI). The status changes directly from Unconfirmed to Resolved (SUR) after which resolution either becomes Worksforme (REW) or Incomplete (REI) before the last resolved stage (CFL).

Table 7.2: Complexity and Fitness Metric of the Spaghetti Model Generated from the entire Event Log as well as the Six Clusters Generated by K-medoid Algorithm using LCS as the Distance Metric

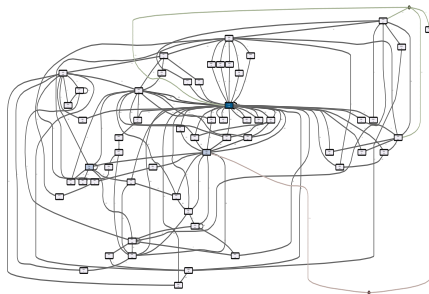
	Cyclomatic Complexity (%age decrease in complexity of clusters)	Fitness
Main Model	143 (-)	0.017
Cluster 1	75 (47.5 %)	0.178
Cluster 2	82 (42.6 %)	0.085
Cluster 3	106 (25.8 %)	0.004
Cluster 4	96 (32.8 %)	0.070
Cluster 5	83 (41.9 %)	0.015
Cluster 6	72 (49.6 %)	0.208

Table 7.3: Complexity and Fitness Metric of the Spaghetti Model Generated from the entire Event Log as well as the Six Clusters Generated by K-medoid Algorithm using DTW as the Distance Metric

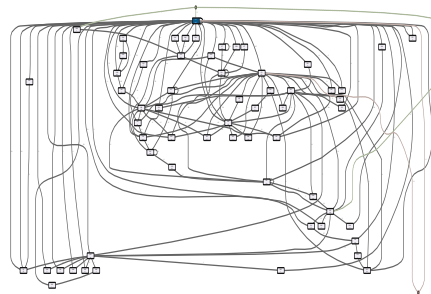
	Cyclomatic Complexity (%age decrease in complexity of clusters)	Fitness
Main Model	143 (-)	0.017
Cluster 1	89 (37.7 %)	0.004
Cluster 2	93 (34.9 %)	0.059
Cluster 3	63 (55.9 %)	0.328
Cluster 4	97 (32.1 %)	0.063
Cluster 5	78 (45.49 %)	0.052
Cluster 6	86 (39.8 %)	0.078



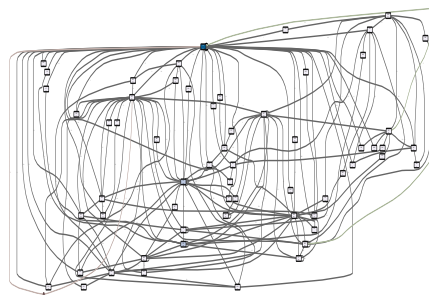
(a) Main Model



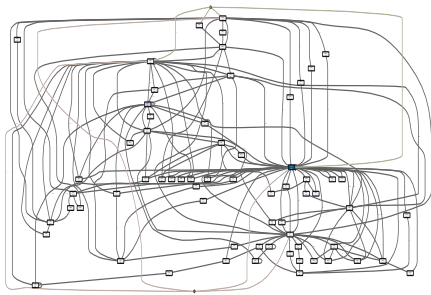
(b) Cluster 1



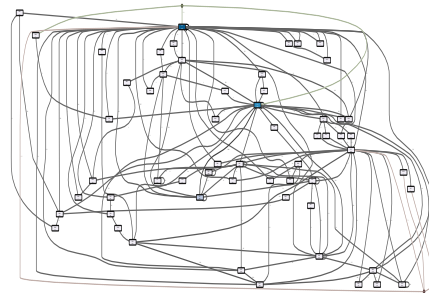
(c) Cluster 2



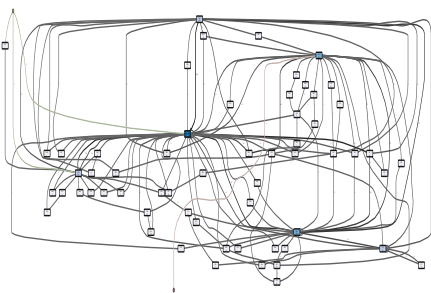
(d) Cluster 3



(e) Cluster 4

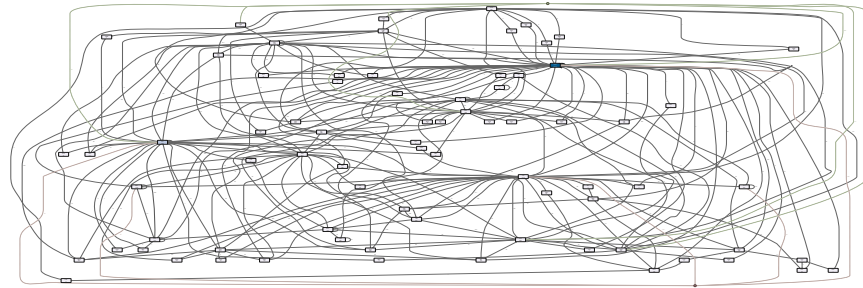


(f) Cluster 5

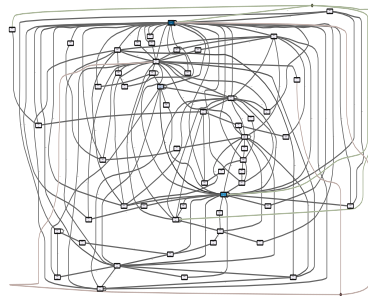


(g) Cluster 6

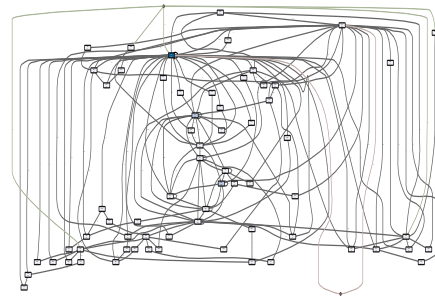
Figure 7.1: Process model generated using Disco from the entire event log (a) and event log of six clusters (b-g) obtained using LCS distance metric.



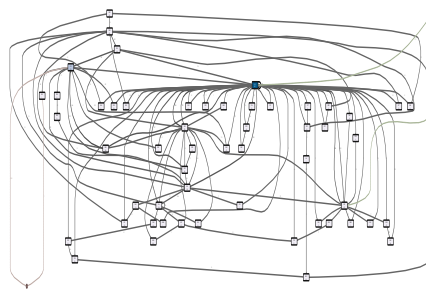
(a) Main Model



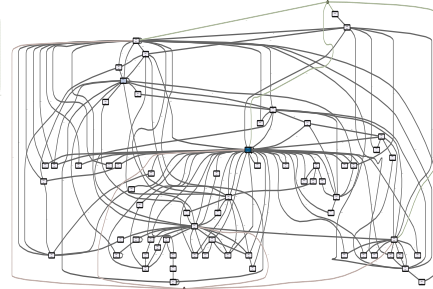
(b) Cluster 1



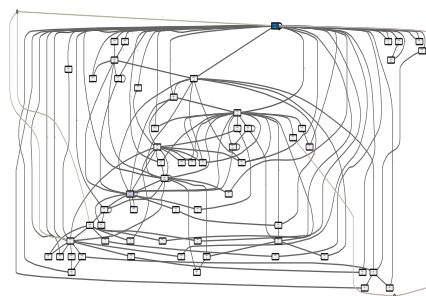
(c) Cluster 2



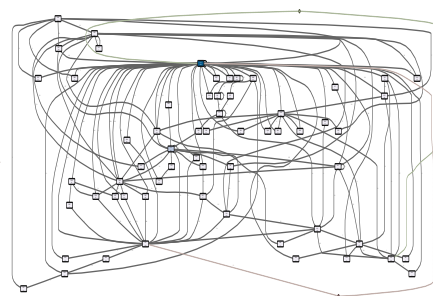
(d) Cluster 3



(e) Cluster 4



(f) Cluster 5



(g) Cluster 6

Figure 7.2: Process model generated using Disco from the entire event log (a) and event log of six clusters (b-g) obtained using DTW distance metric.

8

Analysis

For all the 6 clusters obtained using LCS distance metric, we analyse and study self-loops, back-and-forth, issue reopen, unique traces, event frequency, activity frequency and bottlenecks to show how clustering facilitates a better understanding. Consumable results, actionable information and valuable insights are extracted from the clusters showing that separating structurally similar traces makes the analysis easier.

8.1 Self-loop Analysis

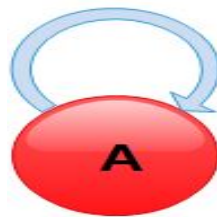


Figure 8.1: Self-Loop

Study of self-loops is important since such loops indicate intensive problems (30). These problems are often difficult to detect because it may seem that at each stage some useful work is being done though actually no progress is being made and the bug is just getting transferred (30). In a process model, a self-loop can be defined as the transition $A \rightarrow A$ i.e. a transition that begins and ends at the same activity (refer Figure 8.1). Self-loops are undesirable and cause redundancy in the bug's trace. Just looking at the count of self-loops of an activity in the event log of spaghetti model is

not enough since it might happen that most of these self-loops are occurring in traces of two or three bugs in which case we cannot generalize and say that this particular activity causes majority of self-loops. Doing self-loop analysis after clustering similar traces helps us to discover if self-loop of an activity appears only in certain kinds of bugs or if it appears in majority in all the clusters. First entry in each cell of Table 8.1 denotes the frequency of self-loop of the activity specified in the first cell of the same row.

1. Self-loop frequency of activity CCC is high in all the six clusters with the count being as high as 3119 in cluster 3. This indicates that many people including users who have no direct role to play in the bug are added in the mailing list. It's an unhealthy practise to repeatedly add/remove people from the mailing list and should be avoided by adding only a few people who are interested in receiving notifications about the bug's progress.
2. Activities ASS and COM has less number of self-loops in all the clusters indicating that most of the time bug is assigned to the right person and correct component type is selected in one go.
3. Many self-loops of activity ATT in clusters 3, 4 & 5 indicates that several properties of attachment file associated with a bug like content-type, description, file-name, isobsolete, flags etc keep on changing and attribute fields are not correctly entered by the user while raising an issue.
4. Many recurrent loops of Activity FLA (Flags) occurs in all clusters except 6. Flags can be of two types: attachment flags and bug flags ¹. Loop involving the former flag indicates that developer has asked other developers to review the code implying that peer code review practice is followed for quite a lot of bugs, while loop involving the latter type indicates that status information of the bug is repeatedly required e.g needinfo flag is set many times sequentially implying that the developer requires more information about the issue raised.
5. High Self-Loop frequency of activity Blocks (BLO) in cluster 3 indicates that several bugs are repeatedly added in the Blocks field which means a lot of bugs

¹<http://www.bugzilla.org/docs/2.22/html/flags-overview.html>

are discovered sequentially which depends upon the current bug. Bugs in this cluster needs to be resolved on a priority basis as several other bugs are dependent on them.

6. Self-loop frequency of activity Depends on (DEP) is extremely high in cluster 3 indicating that several bugs are identified many a time on which the current bug is dependent. It is interesting to note that self-loop frequency of BLO was also high in this cluster indicating that bugs in these clusters are either dependents or dependees.
7. There are 44 self-loops of KEY in cluster 5 indicating that keywords keep on getting updated to identify and group bugs easily.
8. There are 32 self-loops of WHI in cluster 5 indicating that information like tags and status fields of the bugs are entered in the text entry box of the bug repeatedly.

Table 8.1: Self Loops and Back-Forth Analysis

Activity	Main Model	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
ASS	28, CCC/15	2, ATT/1	5, CCC/1	8, CCC/6	8, QAC/3	5, CCC/7	-,-
ATT	266, FLA/116	24, FLA/6	20, FLA/6	102, FLA/40	48, FLA/18	70, FLA/43	2, FLA/3
BLO	152, CCC/39	4, CCC/3	18, CCC/4	72, DEP/18	20, CCC/4	38, CCC/11	-, CCC/2
CCC	6776, FLA/250	524, SNR/60	875, WHI/37	3119, DEP/141	1345, SNR/151	871, FLA/60	42, SUR/53
CFB	26, CCC/16	1, CCC/1	6, CCC/6	11, CCC/6	-,-	4, CCC/3	4,-
CFC	1,-	-,-	1,-	-,-	-,-	-,-	-,-
CFS	151, CFT/14	66, CFT/6	4,-	14, SRV/1	-,-	66, CFT/8	1,-
CFT	38, CCC/15	8, CFS/3	-, CCC/1	-, CCC/2	1,-	29, CCC/10	-,-
COM	2, QAC/3	-,-	-, QAC/1	1, QAC/1	1, QAC/1	-, CCC/1	-,-
DEP	704, CCC/110	9, SNR/2	21, CCC/3	576, CCC/83	41, CCC/6	57, CCC/17	-, CFL/1
FLA	1612, ATT/464	101, CCC/32	93, ATT/25	614, ATT/168	228, ATT/48	557, ATT/186	19, ATT/12
ISC	1, SNU/1	-, SNU/1	-,-	1, SUN/1	-,-	-, SUA/1	-,-
KEY	96, CCC/18	20, CCC/4	6, WHI/1	19, FLA/3	3, VBR/1	44, CCC/10	4, CCC/2
OPS	1, PLA/3	-,-	-, PLA/1	-,-	1,-	-, PLA/2	-,-
PRI	3, TAR/2	-, CCC/1	-, TAR/2	-,-	-,-	1,-	2,-
PRO	3, COM/1	-,-	1,-	-, COM/1	2,-	-,-	-,-
QAC	9, ASS/7	3, ASS/1	-, COM/1	1, ASS/2	5, ASS/4	1,-	-,-
RES	1, SRU/2	1,-	-, SRU/1	-,-	-,-	-,-	-, SRU/1
SEE	3,-	-,-	-,-	1,-	1,-	1,-	-,-
SRR	-, RFF/6	-, RFF/1	-,-	-, RFF/3	-, RES/3	-, RFF/2	-, RES/1
SUM	15, CCC/7	1, CCC/2	3, CCC/1	5, CCC/1	2, CCC/3	4, ASS/1	-,-
TAR	21, CCC/6	-, CCC/1	4, CCC/1	12, CCC/2	1, FLA/2	4, CCC/1	-, ASS/1
VER	6, CCC/10	2,-	-, CCC/3	1, CCC/2	-, PRO/1	1,-	2, CCC/5
WHI	71, CCC/49	5, CCC/1	4, CCC/5	21, CCC/16	4, WHI/5	32, CCC/12	5, CCC/10

8.2 Back-Forth Analysis

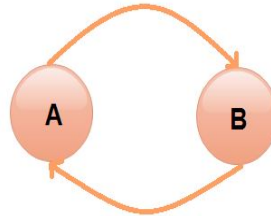


Figure 8.2: Back-Forth

A back-forth loop, also known as ping pong pattern, can be defined as a transition $A \rightarrow B \rightarrow A$ i.e. a transition which begins at activity A, goes to activity B and again ends at A (refer Figure 8.2). Second entry in each cell of Table 8.1 contains the activity with which the activity specified in the first cell of the same row is forming a back-forth loop maximum number of times and also the frequency of that loop. An activity A can be in a back-forth loop with multiple activities e.g. $A \rightarrow B \rightarrow A$ with frequency f_1 and $A \rightarrow C \rightarrow A$ with frequency f_2 with $f_2 > f_1$. Activity C is specified as the second entry in each cell of Table 8.1.

1. Table 8.1 clearly shows that some activities going in back-forth loop with a very high frequency in main model are also depicting the same behavior in all the clusters e.g. ATT looping 116 times with FLA is presenting the exact same pattern in all the clusters.
2. Ping pong patterns that include activity Status Resolved Reopened (SRR) are present in small numbers but are of major interest. The resolve-reopen loop is a problematic pattern. In clusters 1, 3 and 5 SRR is looping with RFF which means that a fixed bug is reopened and again fixed. It can happen when some people who are working to resolve the bug think that the bug has not been properly resolved. Such loops are undesirable because the average time to resolve a re-opened bug can be twice as long as the time to resolve a non re-opened bug (31).
3. Activity DEP forms a back-forth loop with CCC 83 times and CCC forms a loop with DEP 141 times in cluster 3 which might be happening because the team solving other bugs (mentioned in Depends on field) needs to be informed about

the bug's progress so that they can be included in the decision making process of the current bug.

4. CCC is involved in back-forth loop with many activities like SNR, WHI, SUR etc. with high frequency which indicates that more people are involved in the decision making process of confirming the bug's status at every step in the bug's life cycle.
5. Important attributes of the bug like component (COM), assigned (ASS), version (VER), operating system (OPS), summary (SUM) and target milestone (TAR) though are involved in less number in the ping pong patterns but indicates that it takes time to conclude about the values of these fields.

8.3 Event Analysis

A bug life cycle is very diverse consisting of various events ranging from a minimum value of 4 to a maximum of 338. While the main model has a total of 37710 events, on an average 6286 events occur in a cluster. Figure 8.3 shows number of events per case in each of the six clusters. Most of the bugs in Cluster 1 and Cluster 2 have number of events in the range 1-20 as even the main model has most of the cases belonging to this span. Bugs in Cluster 3 have events of varying magnitude in their life-cycle. While almost 20% of bugs have events in range 1-30, there are also 10% of bugs with event length between 101-200. The fewer count of bugs having high event span not easily visible in the main model can be seen in the clusters. Cluster 4 and Cluster 5 have a good percentage of bugs (almost 80-90%) with event cycle length of 1-40, even the main model has very low percentage of bugs above this event span. More than 75% bugs in Cluster 6 have events in the range 1-10 in their life-cycle indicating categorization of bugs of uniform length in the cluster.

8.4 Activity Frequency Analysis

Table 8.2 contains Relative frequency percentage and absolute frequency of all activities present in the clusters. Figure 8.4 shows percentage of cases having some frequently occurring activities in their trace. The high percentage of CFL and CCC ($\approx 100\%$) in all the clusters shows that these activities are present in the trace of almost every

8.4 Activity Frequency Analysis

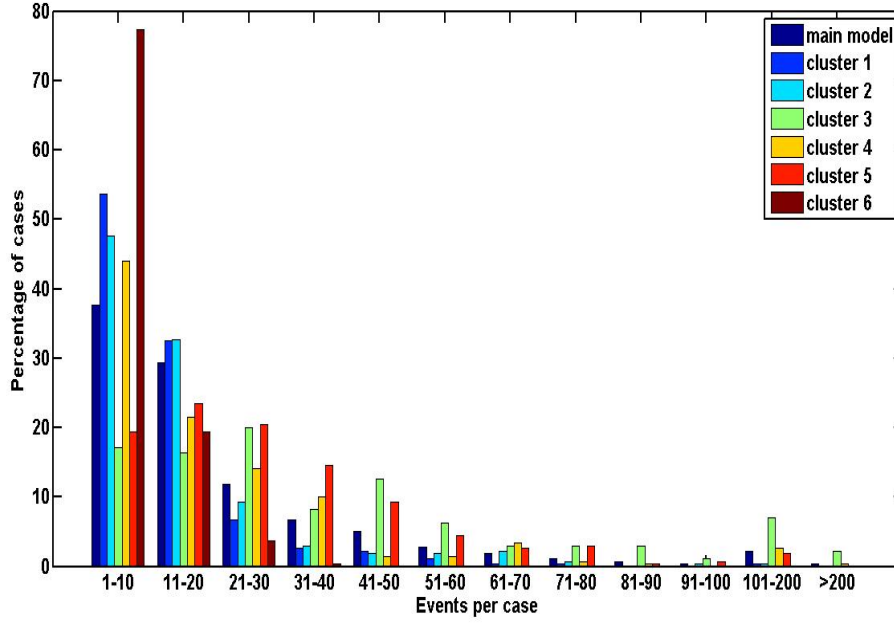


Figure 8.3: No. of Events per Case in each of the six Clusters.

closed bug. We can conclude from the figure that the status of most of the bugs was changed to Resolved from either New (SNR) or Unconfirmed (SUR) status without having multiple assignments (ASS), therefore, less percentage of cases contain ASS. Equal percentages of activities ISC and SUN indicates that bugs that were initially Unconfirmed are Confirmed (ISC) later and set to New state (SUN). Resolution is mostly set to Fixed, Worksforme and Wontfix. Presence of activity BLO in all clusters shows that high dependencies exists among the bugs. Existence of Target Milestone (TAR) in all the clusters in substantial amount depicts that most of the bugs have a predetermined soft deadline upto which it is expected to be fixed.

Table 8.2: Absolute Frequency (Abs) and Relative Frequency Percentage (Rel freq.) for all Activities present in the Clusters.

Activity	Cluster 1 Rel freq. (Abs)	Cluster 2 Rel freq. (Abs)	Cluster 3 Rel freq. (Abs)	Cluster 4 Rel freq. (Abs)	Cluster 5 Rel freq. (Abs)	Cluster 6 Rel freq. (Abs)
CCC	1245 (33.95)	1867 (40.53)	5235 (42.24)	2684 (39.03)	2158 (27.74)	588 (24.58)
FLA	293 (7.99)	280 (6.08)	1564 (12.62)	625 (9.15)	1498 (19.25)	121 (5.06)

8.4 Activity Frequency Analysis

CFL	277 (7.55)	287 (6.23)	317 (2.56)	319 (4.64)	293 (3.77)	286 (11.96)
SNR	170 (4.64)	138 (3)	150 (1.21)	254 (3.69)	108 (1.39)	38 (1.59)
REW	155 (4.23)	79 (1.71)	51 (0.41)	113 (1.64)	16 (0.21)	75 (3.14)
CFS	141 (3.85)	11 (0.24)	42 (0.34)	2 (0.03)	224 (2.88)	4 (0.17)
ASS	106 (2.89)	88 (1.91)	290 (2.34)	373 (5.42)	257 (3.3)	18 (0.75)
KEY	106 (2.89)	119 (2.58)	214 (1.73)	74 (1.08)	245 (3.15)	39 (1.63)
WHI	104 (2.84)	202 (4.38)	281 (2.27)	109 (1.59)	311 (4)	280 (11.71)
VER	84 (2.29)	124 (2.69)	119 (0.96)	71 (1.03)	67 (0.86)	184 (7.69)
ATT	79 (2.15)	90 (1.95)	594 (4.79)	202 (2.94)	487 (6.26)	21 (0.88)
SUR	78 (2.13)	129 (2.8)	47 (0.38)	45 (0.65)	32 (0.41)	234 (9.78)
ISC	76 (2.07)	52 (1.13)	82 (0.66)	139 (2.02)	34 (0.44)	16 (0.67)
SUN	72 (1.96)	46 (1)	71 (0.57)	133 (1.93)	25 (0.32)	11 (0.46)
TAR	57 (1.55)	63 (1.37)	209 (1.69)	95 (1.38)	213 (2.74)	24 (1)
SUM	57 (1.55)	86 (1.87)	156 (1.26)	150 (2.18)	72 (0.93)	24 (1)
BLO	52 (1.42)	138 (3)	488 (3.94)	124 (1.8)	281 (3.61)	18 (0.75)
REF	51 (1.39)	31 (0.67)	136 (1.1)	25 (0.36)	224 (2.88)	26 (1.09)
DEP	48 (1.31)	111 (2.41)	1128 (9.1)	175 (2.55)	236 (3.03)	6 (0.25)
QAC	44 (1.2)	55 (1.19)	105 (0.85)	299 (4.35)	60 (0.77)	13 (0.54)
REE	-	4 (0.09)	-	2 (0.03)	2 (0.03)	2 (0.08)
COM	38 (1.04)	62 (1.35)	93 (0.75)	109 (1.59)	62 (0.8)	21 (0.88)
REI	18 (0.49)	37 (0.8)	16 (0.13)	9 (0.13)	4 (0.05)	113 (4.72)
CFT	37 (1.01)	3 (0.07)	14 (0.11)	2 (0.03)	127 (1.63)	-
RED	33 (0.9)	40 (0.87)	28 (0.23)	33 (0.48)	20 (0.26)	23 (0.96)
OPS	29 (0.79)	43 (0.93)	85 (0.69)	82 (1.19)	56 (0.72)	7 (0.29)
SAR	28 (0.76)	17 (0.37)	89 (0.72)	14 (0.2)	136 (1.75)	9 (0.38)
SNA	23 (0.63)	17 (0.37)	104 (0.84)	41 (0.6)	125 (1.61)	3 (0.13)
PLA	22 (0.6)	37 (0.8)	79 (0.64)	70 (1.02)	56 (0.72)	8 (0.33)
CFB	14 (0.38)	85 (1.85)	154 (1.24)	-	91 (1.17)	6 (0.25)
SRV	14 (0.38)	21 (0.46)	42 (0.34)	27 (0.39)	88 (1.13)	4 (0.17)
SNE	10 (0.27)	6 (0.13)	11 (0.09)	24 (0.35)	2 (0.03)	3 (0.13)
PRI	10 (0.27)	32 (0.69)	41 (0.33)	46 (0.67)	12 (0.15)	15 (0.63)
RES	9 (0.25)	26 (0.56)	30 (0.24)	48 (0.7)	5 (0.06)	21 (0.88)
SNM	6 (0.16)	2 (0.04)	8 (0.06)	12 (0.17)	3 (0.04)	2 (0.08)
PRO	6 (0.16)	12 (0.26)	22 (0.18)	26 (0.38)	15 (0.19)	1 (0.04)
SRR	6 (0.16)	8 (0.17)	65 (0.52)	27 (0.39)	37 (0.48)	12 (0.5)
SRU	5 (0.14)	12 (0.26)	12 (0.1)	22 (0.32)	1 (0.01)	10 (0.42)
SCM	5 (0.14)	4 (0.09)	1 (0.01)	2 (0.03)	-	3 (0.13)
SEE	5 (0.14)	5 (0.11)	10 (0.08)	6 (0.09)	9 (0.12)	3 (0.13)
URL	4 (0.11)	3 (0.07)	28 (0.23)	17 (0.25)	3 (0.04)	12 (0.5)
CFC	4 (0.11)	2 (0.04)	1 (0.01)	1 (0.01)	-	-
RFF	4 (0.11)	2 (0.04)	23 (0.19)	8 (0.12)	18 (0.23)	2 (0.08)

8.4 Activity Frequency Analysis

RFW	-	3 (0.07)	4 (0.03)	-	1 (0.01)	1 (0.04)
GRO	3 (0.08)	8 (0.17)	4 (0.03)	4 (0.06)	-	9 (0.38)
SNU	3 (0.08)	3 (0.07)	1 (0.01)	1 (0.01)	1 (0.01)	1 (0.04)
SEN	2 (0.05)	1 (0.02)	2 (0.02)	5 (0.07)	1 (0.01)	2 (0.08)
STE	2 (0.05)	-	2 (0.02)	1 (0.01)	-	1 (0.04)
SMN	2 (0.05)	3 (0.07)	1 (0.01)	5 (0.07)	-	1 (0.04)
SNT	2 (0.05)	-	2 (0.02)	3 (0.04)	2 (0.03)	-
RFD	-	-	-	-	1 (0.01)	-
SAN	2 (0.05)	2 (0.04)	34 (0.27)	29 (0.42)	15 (0.19)	-
SCN	-	2 (0.04)	-	1 (0.01)	-	8 (0.33)
SBM	-	1 (0.02)	-	-	-	-
SEM	-	1 (0.02)	1 (0.01)	3 (0.04)	-	-
SME	-	1 (0.02)	-	7 (0.1)	-	1 (0.04)
SMM	-	1 (0.02)	-	1 (0.01)	-	-
SMC	1 (0.03)	1 (0.02)	-	-	1 (0.01)	-
SNC	1 (0.03)	4 (0.09)	2 (0.02)	1 (0.01)	-	3 (0.13)
SUA	1 (0.03)	1 (0.02)	5 (0.04)	1 (0.01)	5 (0.06)	2 (0.08)
SVU	-	1 (0.02)	-	6 (0.09)	-	1 (0.04)
SRN	1 (0.03)	2 (0.04)	1 (0.01)	15 (0.22)	4 (0.05)	2 (0.08)
SNB	1 (0.03)	-	-	-	1 (0.01)	-
SRA	1 (0.03)	-	4 (0.03)	1 (0.01)	2 (0.03)	-
ALI	-	-	4 (0.03)	4 (0.06)	-	2 (0.08)
SVR	-	-	3 (0.02)	3 (0.04)	2 (0.03)	-
RFI	-	-	2 (0.02)	-	1 (0.01)	-
STM	-	-	1 (0.01)	1 (0.01)	1 (0.01)	1 (0.04)
STN	-	-	-	2 (0.03)	-	-
SET	-	-	-	1 (0.01)	-	-
SBN	-	-	-	1 (0.01)	1 (0.01)	2 (0.08)
SMT	-	-	-	-	2 (0.03)	-
SCE	-	-	-	-	-	1 (0.04)
SME	-	-	-	-	-	1 (0.04)
REX	67 (1.83)	70 (1.52)	63 (0.51)	107 (1.56)	12 (0.15)	19 (0.79)
REN	20 (0.55)	26 (0.56)	23 (0.19)	30 (0.44)	14 (0.18)	28 (1.17)
SEC	-	-	-	-	-	1 (0.04)

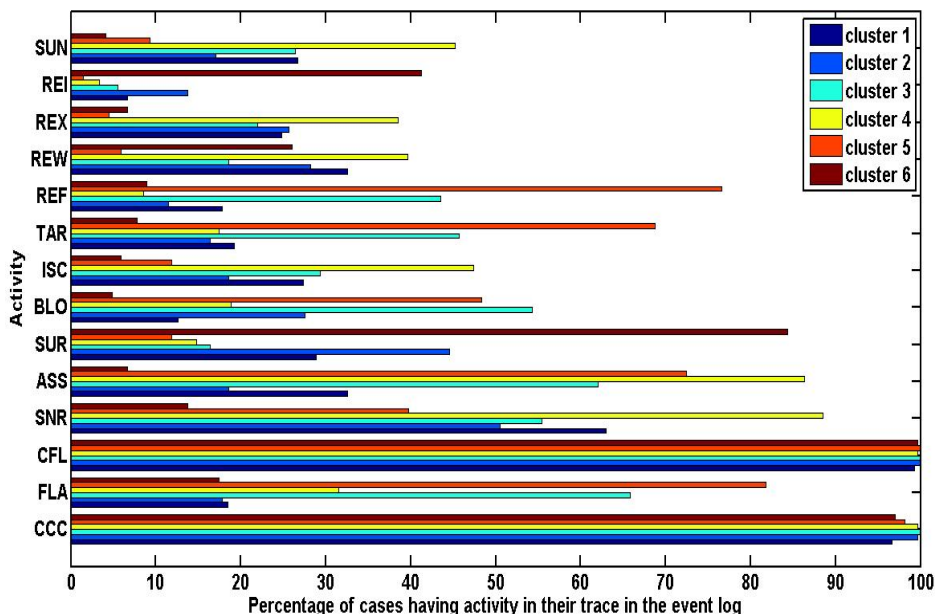


Figure 8.4: Distribution of some Frequently Occurring Activities over the cases

8.5 Reopen Analysis

Bug reopening is equally important in open source systems like Bugzilla as it is in closed source or commercial systems (32). It increases the costs of maintaining the software, lessens the user-perceived quality of the system and leads to extra and needless rework by already loaded developers (31). Reopening of a substantial number of fixed bugs can indicate instability in the system (32). Analysis of factors leading to bug reopening will help in improving the quality of bug fixing process and countering all these problems.

We are taking into account the following factors (22) (31) (32) that contribute in reopening of bugs:

1. Verified: A bug verified by a Quality Assurance agent may get reopened if some useful information about the bug becomes available that demands to have it reviewed again.
2. Fixed: A fixed bug may have its reopening if the fix proposed seems to have faults and is not complete and entirely correct solution.

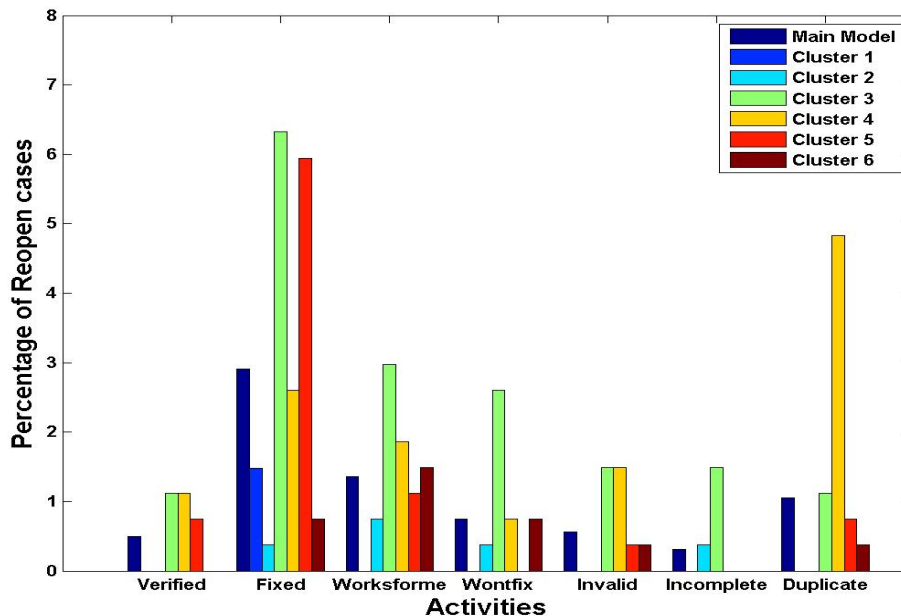


Figure 8.5: Reopen Analysis.

3. Duplicate: If the bug is not studied deeply and few of its symptoms match with some already existing bug, it is incorrectly assumed to be the case of duplicacy.
4. Wontfix/Invalid/Incomplete/Worksforme: There are high chances of re-openings if earlier the bug was not been able to fix (Wontfix), it was not categorized as a bug (Invalid), it was reported with incomplete information (Incomplete) or if it was not successfully reproduced (Worksforme).

Clustering helps in analysing whether the reopening due to an activity is happening globally throughout the main model or in a certain set of similar bugs.

1. Percentage of re-openings due to Verified (SRV) and Invalid (REN) resolutions is nearly same in main model, still reopening after bug verification (SRV) is supported by 3 clusters while REN appears in 4 clusters suggesting that former appears in similar characteristic bugs while the latter can appear in all kinds of bugs. Also less percentage of SRV is supported by the fact that they are verified by a Quality Assurance agent (QAC) who confirms that a proper fix has been achieved.

2. In almost all clusters a good amount of re-openings occur after resolution of bug was set as Fixed (REF) signifying that resolution set as Fixed before the reopening is uniformly distributed across all the bugs in good amount thus appearing in all clusters. It also indicates bad understanding and management in fixing the bugs at first, leading to loss of time in analysing and correcting the same bug again. Reopening after resolution was set to Worksforme (REW) is half in percentage than REF in the main model, still 5 out of 6 clusters contain reopening after REW indicating that this resolution is not limited to some closely related bugs but rather appears at random.
3. Reopening due to the activity Worksforme (REW) is also contributed by 5 out of 6 clusters suggesting that bugs entering into the system are initially difficult to reproduce, thus are left for future references/information using which they will be reopened again.
4. Wontfix, Invalid, Duplicate factors cause reopening in 4 clusters. Invalid reopening is half in percentage than duplicate reopening in the main model and their presence in equal numbers in clusters signifies that invalid label could not be categorized under a category of clusters having certain common properties and is spread randomly in the data.

The description of the clusters mentioned in Table 7.1 also support the kind of activities leading to the reopening in various clusters.

1. Cluster 1 contributes only to reopening after bug is Fixed (REF) which can also be seen from the description of this cluster. Most of the bugs in this cluster are as described confirmed to be true, resolved and await for the verification instead of being verified and thus are later reopened.
2. In Cluster 2 and Cluster 6 the maximum percentage of reopening is due to factor Worksforme (REW) informing that the bugs are unable to be reproduced and hence in the cluster description we see various flags, attachments and dependencies being added to the bugs to make its definition clear.
3. In Cluster 3 and Cluster 5 reopening after REF is hugely dominant as for most of the bugs resolution was set as Fixed, but later we see clusters containing various assignments, attachments, dependencies, keywords, summaries being defined

indicating that the fix was not absolute and it requires reopening and looking again.

4. Cluster 4 has most of the cases of the bugs being marked duplicate incidentally and reopened thereafter. In this cluster we see a lot of unconfirmed bugs being changed to Resolved State (SUR) indicating that the bugs are not properly examined before their resolution is set. The reason for not confirming the identity of the bug is that they are considered as duplicates of existing bugs.

8.6 Unique Traces

A trace is a sequence of activities appearing in chronological order of occurrence. For analysis we consider unique traces and their frequency of occurrence in data set. The main model consists of 1513 variants (unique sequences of activities). For looking at the distribution of similar traces in the clusters, a few topmost occurring variants are considered. Since clustering groups similar objects together, similar traces should go in the same cluster. The experimental results show that cases that follow the same variant appear in the same cluster. As can be seen from Table 8.3, the top 4 variants of main model goes in Cluster 6 while the 5th one lies in Cluster 1.

8.7 Bottleneck Identification

Bottleneck refers to those components of process model that consume comparatively more time than the rest of the system causing the entire process to slow down. Identification of principal factors constraining the process speed can help a process analyst in working upon the causes that deter the performance of a process. We compute the mean time taken for every transition between two activities in both the main model as well as in all the clusters to conclude as to how clustering helps in analysis of bottlenecks in a better manner. For analysis, we consider the transitions taking the maximum amount of time i.e. discovering the largest bottlenecks present in the models.

1. The percentage of bottlenecks taking mean time more than 500 and 1000 days in all the models is shown in Figure 8.6. From the Figure we can observe that

8.7 Bottleneck Identification

Table 8.3: Most Occurring Variants of Main Model and each of the Six Clusters with their Frequency of Occurrence

Cluster	Variant Freq. (%age)	Trace
Main Model	12 (0.74)	VER → CCC → SUR → CCC → REW → CFL
	11 (0.68)	VER → WHI → SUR → CCC → REI → WHI → CFL
	7 (0.43)	SUR → RED → CCC → CFL → CCC
	7 (0.43)	VER → CCC → SUR → CCC → REI → CFL
	5 (0.31)	VER → CCC → CCC → SUR → CCC → REW → CFL
Cluster 1	5 (1.85)	VER → CCC → CCC → SUR → CCC → REW → CFL
	3 (1.11)	SUR → CCC → REW → CFL
	3 (1.11)	SNR → CCC → REW → CFL
	2 (0.74)	CCC → CCC → SNR → REW → CFL
Cluster 2	2 (0.74)	CCC → CCC → VER → CCC → CCC → SUR → REW → CFL
	2 (0.74)	CCC → SNR → RED → CFL → CCC
	2 (0.74)	SUR → RED → CFL → CCC
	2 (0.74)	WHI → SUR → REI → CFL → CCC → CCC
Cluster 3	2 (0.74)	VER → CCC → CCC → FLA → SUR → REN → FLA → CFL
	1 (0.37)	SUM → SUR → REX → CFL → SRU → RES → SUR → REX → CFL → CCC → CCC → QAC → CCC → CCC
	1 (0.37)	CCC → FLA → FLA → SUR → REW → CFL
	1 (0.37)	SUM → PLA → OPS → CCC → CCC → CCC → CCC → SNR → CCC → REX → CFL
Cluster 4	1 (0.37)	SUM → ASS → SNR → CCC → REW → CFL → SUM → PLA
	1 (0.37)	SUR → REW → CFL → QAC → CCC → SUM → SUN
	1 (0.37)	SUM → SUR → REX → CFL → CCC → SRV → CCC
	1 (0.37)	CCC → REW → CFL → CCC → CCC
Cluster 5	2 (0.74)	SUR → REN → CCC → CFL → SRV → CCC
	2 (0.74)	CCC → CCC → SUR → REN → CCC → CFL → CCC
	2 (0.74)	SNR → RED → CFL → CCC
	1 (0.37)	QAC → ASS → SUR → REW → CCC → CFL → CCC
Cluster 6	12 (4.46)	VER → CCC → SUR → CCC → REW → CFL
	11 (4.09)	VER → WHI → SUR → CCC → REI → WHI → CFL
	7 (2.6)	SUR → RED → CCC → CFL → CCC
	7 (2.6)	VER → CCC → SUR → CCC → REI → CFL

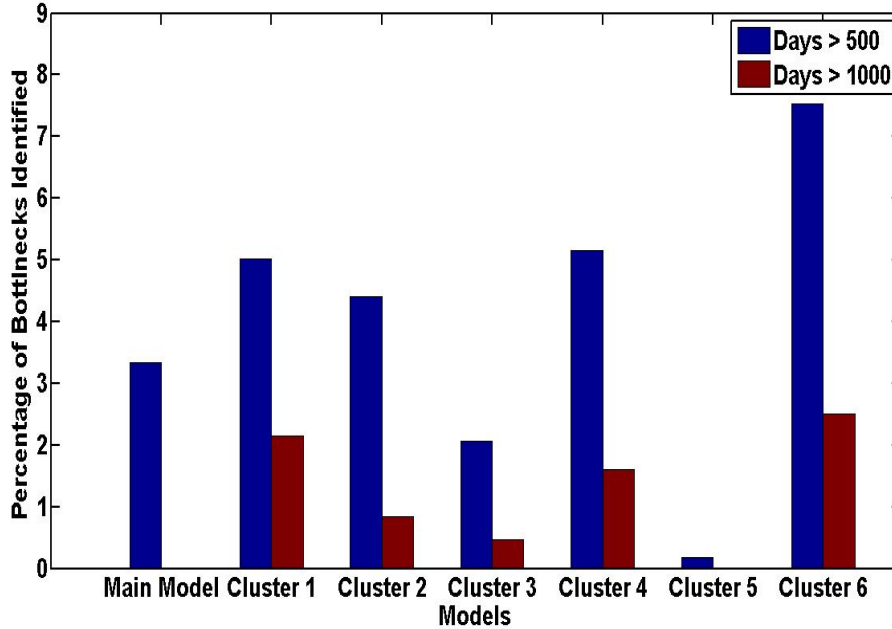


Figure 8.6: Bottleneck Analysis.

percentage of bottlenecks with period more than 500 days (mean value) is greater in 4 clusters as compared to the main model.

While for duration greater than 1000 days (mean value) each cluster has strictly higher percentage count than in the main model. Rather one cannot identify any bottleneck transitions that take more than 1000 days in the main model. It is due to absolute count of transitions which is less in cluster than in main model producing greater mean value for the clusters. Thus bottlenecks that are not quite evident in the main model are clearly visible in the clusters.

2. Set of transactions taking mean time greater than 1000 days found in both the main model as well as clusters are:
 - (a) $SRV \rightarrow CFB$, $SRV \rightarrow QAC$ implying that after a bug is verified (SRV), there is a large gap before any other actions like contacting another Quality Assurance agent (QAC), making any custom extension field changes (CFB) in Bugzilla are done. This indicates that a once a bug is verified it is not acted upon much.

- (b) ISC \rightarrow SNR, suggesting huge delay between the time a bug is confirmed to be true (ISC) to the time appropriate actions are taken to resolve it (SNR). It indicates that in some cases it takes a lot of time to understand and confirm that the issue raised is actually a bug.
 - (c) CFL \rightarrow SEE informing that over 1000 days can be taken for bugs that are yet not verified from their last resolved stage (CFL) (probably due to resolution found to be Invalid, Incomplete, Duplicate, Wontfix or Worksforme) to the time another bug is found in some other installation related to it and is thus linked with for reference (SEE).
3. The bottlenecks found in clusters (not observed in main model) taking mean time greater than 1000 days are:
- (a) Some of the activities performed before setting the Resolution from New state (SNR) like ASS \rightarrow SNR, ATT \rightarrow SNR, CFB \rightarrow SNR, TAR \rightarrow SNR span over 1200 days suggesting it takes years of time to finally resolve a bug even after it is properly assigned (ASS), has its associated attachments (ATT) for references, custom extension fields are specified in Bugzilla (CFB) or even if target milestone is set.
 - (b) Transitions QAC \rightarrow SUN, QAC \rightarrow SUR, QAC \rightarrow SNR take more than 3 years to execute indicating that it can take years for a bug to change its status to New or Resolved even after contacting the Quality Assurance (QAC).
 - (c) CCC \rightarrow REF, CCC \rightarrow REW, CCC \rightarrow REX. It takes more than 3 years to come to the decision of setting the resolution of bug to Fixed (REF), Worksforme (REW) or Wontfix (REX). It shows that deciding the appropriate resolution of the bug takes years of time even after appropriate users/developers are involved and informed about the bug progress.
 - (d) Changing the status of bug directly to Resolved from Unconfirmed without assigning it the status of New may require a period of 4 years (transitions SUM \rightarrow SUR, OPS \rightarrow SUR) even after the important attributes of bug like summary (SUM) and operating system (OPS) are properly defined.

8.7 Bottleneck Identification

- (e) QAC → SNR takes more than 3 years to execute indicating that it can take years for a bug to change its status to New or Resolved even after contacting the Quality Assurance (QAC) who verifies that the issue raised is a bug.

9

Automate Clustering to Determine the Best Cluster Solution

Clustering can give many different solutions depending upon the algorithm used, initial cluster centers/medoids chosen, number of iterations and number of clusters specified. Out of the many possible solutions, we should select the one where clusters have low complexity and high fitness value for enabling better analysis. In this section we propose an algorithm that automatically computes the goodness of process models and returns the best cluster set for analysis. The algorithm takes as input an event log and returns a set of clusters that maximizes the objective function which is to reduce complexity and increase fitness. We compute the goodness ratio (G_Ratio) by dividing the weighted average of fitness values (F_score) by the weighted average of complexity values (C_score) of all clusters present in the given solution set and returns the set having maximum G_Ratio. A weighted average of a cluster is the average taken with respect to number of traces present in the cluster. Since clustering distributes the traces non uniformly in clusters, fitness and complexity contribution by each cluster has to be taken with respect to the absolute count of cases present in that cluster. The minimum value of G_Ratio is 0 (when each cluster has fitness 0 making F_score = 0). To validate the automated clustering algorithm, the experimental dataset described in Table 4.1 was split into four equal sub datasets and each subset was experimented with the proposed algorithm using LCS similarity metric. Algorithm 9 runs the clustering

algorithm multiple times over the input event log to select the best cluster set. So if a process analyst wants to compare the goodness of two or more clustering solutions, he can use our proposed approach. For the given dataset we have run the clustering algorithm thrice. The motivation behind running the algorithm thrice over same Event-Log is to determine how much does the goodness of a process model varies with respect to initial random selection of the medoids in the k medoid clustering algorithm. Thus, clustering was repeated thrice by the automated clustering algorithm to get an estimation as to how much the deviations of fitness and complexity occur with each new clustering run. Table 9.1 gives the G_Ratio of all the three iterations performed on all four sub datasets as well as the iteration whose solution set is determined to be the best by our proposed algorithm.

Table 9.1: Automated Clustering Algorithm Analysis

Dataset No.	Iteration No.	Weighted Complexity	Weighted Fitness	G_Ratio	Result
1	1	90.98	0.190	2.08×10^{-03}	-
1	2	92.38	0.158	1.71×10^{-03}	-
1	3	90.91	0.227	2.49×10^{-03}	Selected
2	1	99.43	0.275	2.08×10^{-03}	-
2	2	100.99	0.205	2.7×10^{-03}	Selected
2	3	105.8	0.213	2.01×10^{-03}	-
3	1	92.05	0.125	1.35×10^{-03}	-
3	2	91.39	0.106	1.15×10^{-03}	-
3	3	93.47	0.218	2.33×10^{-03}	Selected
4	1	81.36	0.394	4.84×10^{-03}	Selected
4	2	85.57	0.270	3.15×10^{-03}	-
4	3	85.40	0.211	2.47×10^{-03}	-

Algorithm 9: Automate Clustering

Data: History data of bugs

Result: Best cluster set

- 1 *Event Log Creation*(*History data of bugs*)
 - 2 *Sequential Data Creation*(*Event Log of Bugs*)
 - 3 generate 3 cluster sets S_1 , S_2 and S_3 by calling
 k *Medoid Clustering*(*Event log in sequential data format*) that uses LCS and DTW similarity for input k value
 - 4 **foreach** cluster set S_i consisting of m clusters **do**
 - 5 **for** $j \leftarrow 1$ **to** m **do**
 - 6 discover process model P_j
 - 7
$$C_score_i = \sum_{j=1}^m \frac{Complexity(Xml\ format\ input\ of\ P_j) * t_j}{t_j}$$
$$F_score_i = \sum_{j=1}^m \frac{Fitness(Xml\ format\ input\ of\ P_j, Event\ log\ in\ sequential\ format\ of\ cluster\ C_j) * t_j}{t_j}$$
$$G_Ratio_i = F_score / C_score$$
 - where t_j is total traces in event log of cluster C_j
 - 8 return the cluster set S_i with the maximum G_Ratio.
-

10

Limitations and Future Work

The experimental dataset is small and the clustering has been done on Event Logs consisting of less process instances. We plan to validate our approach on larger datasets.

The experimental dataset has been taken from a single Issue tracking System. The presented work can be applied on larger and diverse datasets. Future work includes application of our proposed technique on various other real-life datasets.

We have adapted the basic version of k medoid algorithm. Not much focus was given on complexity and running time of the algorithm. As a part of future work, we would focus on these two aspects of the algorithms.

K medoid algorithm is known to be sensitive to initial cluster center assignment. We plan to work on an algorithm that can select the traces which can be the best candidates for initial medoid assignment.

In our proposed technique the analyst has to provide number of clusters (k) as input. Selecting the value of k is a popular research problem for which lot of work has been done and continues to be done.

We plan to experiment our proposed distance metrics with other clustering algorithms and compare their performances. We also plan to compare the performance of our two proposed distance metrics and to find out which metric works best and when.

Conclusion

Analysing the results of mining real world unstructured event logs that show adhoc behavior is difficult due to production of complex spaghetti- like process models. Our work is a contribution towards simplifying these complex models by means of clustering so that they can be easily understood by the process analyst. Fitness and structural complexity of the process models is improved by forming groups of structurally similar traces which makes the model easier to comprehend. K medoid algorithm has been adapted using two different distance metrics- LCS and DTW to obtain clusters having good intra-class similarity. K medoid is an efficient clustering algorithm which is not sensitive to outliers and noisy data. We have demonstrated the effectiveness of our proposed technique by performing a real life case study on Firefox browser project. We have successfully shown that clustering enables better analysis, making it easier to identify bottlenecks, study reopening of bugs, self & back forth loops. An algorithm has been proposed to automate the clustering process which has been tested on four different datasets. It returns the cluster set with highest goodness ratio from the various cluster sets obtained.

References

- [1] WIL VAN DER AALST. **Process Mining: Overview and Opportunities**. *ACM Trans. Manage. Inf. Syst.*, **3**(2):7:1–7:17, July 2012. vi, 6
- [2] ALAIN APRIL AND ALAIN ABRAN. *Software Maintenance Management: Evaluation and Continuous Improvement (Practitioners)*. Wiley-IEEE Computer Society Pr, 2008. 2
- [3] THOMAS M. PIGOSKI. *Practical Software Maintenance: Best Practices for Managing Your Software Investment*. John Wiley & Sons, Inc., New York, NY, USA, 1996. 2
- [4] *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. 2
- [5] WIL M. AALST. **Transactions on Petri Nets and Other Models of Concurrency II**. chapter Process-Aware Information Systems: Lessons to Be Learned from Process Mining, pages 1–26. Springer-Verlag, Berlin, Heidelberg, 2009. 6
- [6] WIL VAN DER AALST. **Process Mining: Making Knowledge Discovery Process Centric**. *SIGKDD Explor. Newsl.*, **13**(2):45–49, May 2012. 7
- [7] W. M. P. VAN DER AALST, H. A. REIJERS, A. J. M. M. WEIJTERS, B. F. VAN DONGEN, A. K. ALVES DE MEDEIROS, M. SONG, AND H. M. W. VERBEEK. **Business Process Mining: An Industrial Application**. *Inf. Syst.*, **32**(5):713–732, July 2007. 7
- [8] S.A. WHITE AND D. MIERS. *BPMN Modeling and Reference Guide: Understanding and Using BPMN*. Future Strategies Incorporated, 2008. 9

-
- [9] REMCO M. DIJKMAN, MARLON DUMAS, AND CHUN OUYANG. **Semantics and Analysis of Business Process Models in BPMN**. *Inf. Softw. Technol.*, **50**(12):1281–1294, November 2008. 9
- [10] CHRISTIAN W. GÜNTHER AND WIL M. P. VAN DER AALST. **Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics**. In *Proceedings of the 5th International Conference on Business Process Management, BPM'07*, pages 328–343, Berlin, Heidelberg, 2007. Springer-Verlag. 13, 18
- [11] WIL M. P. VAN DER AALST AND CHRISTIAN W. GÜNTHER. **Finding Structure in Unstructured Processes: The Case for Process Mining**. In *Seventh International Conference on Application of Concurrency to System Design (ACSD 2007), 10-13 July 2007, Bratislava, Slovak Republic*, pages 3–12, 2007. 13, 14
- [12] GABRIEL M. VEIGA AND DIOGO R. FERREIRA. **Understanding Spaghetti Models with Sequence Clustering for ProM**. In *Business Process Management Workshops, BPM 2009 International Workshops, Ulm, Germany, September 7, 2009. Revised Papers*, **43** of *Lecture Notes in Business Information Processing*, pages 92–103. Springer, 2010. 13, 14
- [13] R. P. JAGADEESH CHANDRA BOSE AND WIL M. P. VAN DER AALST. **Context Aware Trace Clustering: Towards Improving Process Mining Results**. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2009, April 30 - May 2, 2009, Sparks, Nevada, USA*, pages 401–412, 2009. 14
- [14] DIOGO FERREIRA, MARIELBA ZACARIAS, MIGUEL MALHEIROS, AND PEDRO FERREIRA. **Approaching Process Mining with Sequence Clustering: Experiments and Findings**. In *Proceedings of the 5th International Conference on Business Process Management, BPM'07*, pages 360–374, Berlin, Heidelberg, 2007. Springer-Verlag. 14
- [15] JOCHEN DE WEERDT, SEPPE K. L. M. VANDEN BROUCKE, JAN VANTHIENEN, AND BART BAESENS. **Active Trace Clustering for Improved Process Discovery**. *IEEE Trans. Knowl. Data Eng.*, **25**(12):2708–2720, 2013. 14

-
- [16] MINSEOK SONG, CHRISTIAN W. GÜNTHER, AND WIL M. P. VAN DER AALST. **Trace Clustering in Process Mining.** In *Business Process Management Workshops, BPM 2008 International Workshops, Milano, Italy, September 1-4, 2008. Revised Papers*, pages 109–120, 2008. 14, 15
- [17] GIANLUIGI GRECO, ANTONELLA GUZZO, LUIGI PONTIERI, AND DOMENICO SACCÀ. **Discovering Expressive Process Models by Clustering Log Traces.** *IEEE Transactions on Knowledge and Data Engineering*, **18**(8):1010–1027, 2006. 15
- [18] ANA KARLA ALVES DE MEDEIROS, ANTONELLA GUZZO, GIANLUIGI GRECO, WIL M. P. VAN DER AALST, A. J. M. M. WEIJTERS, BOUDEWIJN F. VAN DONGEN, AND DOMENICO SACCÀ. **Process Mining Based on Clustering: A Quest for Precision.** In *Business Process Management Workshops, BPM 2007 International Workshops, BPI, BPD, CBP, ProHealth, RefMod, semantics4ws, Brisbane, Australia, September 24, 2007, Revised Selected Papers*, pages 17–29, 2007. 15
- [19] W.M.P.VANDER AALST. **Business alignment: using process mining as a tool for Delta analysis and conformance testing.** *Requirements Engineering*, **10**(3):198–211, 2005. 15
- [20] RAFAEL ACCORSI AND THOMAS STOCKER. **On the Exploitation of Process Mining for Security Audits: The Conformance Checking Case.** In *Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC '12*, pages 1709–1716, New York, NY, USA, 2012. ACM. 15
- [21] A. ROZINAT AND W.M.P. VAN DER AALST. **Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models.** In M. CASTELLANOS AND T. WEIJTERS, editors, *First International Workshop on Business Process Intelligence (BPI'05)*, pages 1–12, Nancy, France, September 2005. 15
- [22] MONIKA GUPTA AND ASHISH SUREKA. **Nirikshan: Mining Bug Report History for Discovering Process Maps, Inefficiencies and Inconsistencies.**

-
- In *Proceedings of the 7th India Software Engineering Conference*, ISEC '14, pages 1:1–1:10, New York, NY, USA, 2014. ACM. 15, 34, 51
- [23] L. KAUFMAN AND P. J. ROUSSEEUW. *Finding groups in data: an introduction to cluster analysis*. John Wiley and Sons, New York, 1990. 27
- [24] ROBERT A. WAGNER AND MICHAEL J. FISCHER. **The String-to-String Correction Problem**. *J. ACM*, **21**(1):168–173, January 1974. 30
- [25] GUOJUN GAN, CHAOQUN MA, AND JIANHONG WU. *Data clustering - theory, algorithms, and applications*. SIAM, 2007. 31
- [26] J. CARDOSO, J. MENDLING, G. NEUMANN, AND H. A. REIJERS. **A Discourse on Complexity of Process Models**. In *Proceedings of the 2006 International Conference on Business Process Management Workshops*, BPM'06, pages 117–128, Berlin, Heidelberg, 2006. Springer-Verlag. 33
- [27] VOLKER GRUHN AND RALF LAUE. **Complexity metrics for business process models**. In *in: W. Abramowicz, H.C. Mayr (Eds.), 9th International Conference on Business Information Systems (BIS 2006), Lecture Notes in Informatics*, pages 1–12. 33
- [28] ANTTI LATVA-KOIVISTO. **Finding a Complexity Measure for Business Process Models**, 2001. 33
- [29] THOMAS J. MCCABE. **A Complexity Measure**. In *Proceedings of the 2Nd International Conference on Software Engineering*, ICSE '76, pages 407–, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press. 33
- [30] CHRISTINE A. HALVERSON, JASON B. ELLIS, CATALINA DANIS, AND WENDY A. KELLOGG. **Designing Task Visualizations to Support the Coordination of Work in Software Development**. In *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, CSCW '06, pages 39–48, New York, NY, USA, 2006. ACM. 42
- [31] EMAD SHIHAB, AKINORI IHARA, YASUTAKA KAMEI, WALID M. IBRAHIM, MASAO OHIRA, BRAM ADAMS, AHMED E. HASSAN, AND KEN ICHI MATSUMOTO.

- Studying Re-opened Bugs in Open Source Software.** 18, pages 1005–1042. Springer, 2013. 46, 51
- [32] THOMAS ZIMMERMANN, NACHIAPPAN NAGAPPAN, PHILIP J. GUO, AND BRENDAN MURPHY. **Characterizing and Predicting Which Bugs Get Re-opened.** In *Proceedings of the 34th International Conference on Software Engineering (ICSE 2012 SEIP Track)*. IEEE, June 2012. 51
- [33] B. F. VAN DONGEN, A. K. A. DE MEDEIROS, H. M. W. VERBEEK, A. J. M. M. WEIJTERS, AND W. M. P. VAN DER AALST. **The Prom Framework: A New Era in Process Mining Tool Support.** In *Proceedings of the 26th International Conference on Applications and Theory of Petri Nets, ICATPN'05*, pages 444–454, Berlin, Heidelberg, 2005. Springer-Verlag.