MIMANSA: Process Mining Software Repositories from Student Projects in an Undergraduate Software Engineering Course

Student Name: MEGHA MITTAL

IIIT-D-MTech-CS-DE-12-043
Jan xx, 2013

Indraprastha Institute of Information Technology
New Delhi

Thesis Committee
Ashish Sureka (Chair)
Rahul Purandare
Sanjay Goel

Submitted in partial fulfillment of the requirements
for the Degree of M.Tech. in Computer Science,
with specialization in Data Engineering

Keywords: Mining Software Repositories, Process Mining, Education Data Mining, Learning Analytic, Software Engineering Education

Certificate

This is to certify that the thesis titled **"MIMANSA: Process Mining Software Repositories from Student Projects in an Undergraduate Software Engineering Course"** submitted by **Megha Mittal** for the partial fulfillment of the requirements for the degree of *Master of Technology* in *Computer Science & Engineering* 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.

Prof. Ashish Sureka
Indraprastha Institute of Information Technology, New Delhi

Abstract

An undergraduate level Software Engineering course generally consists of a team-based semester long project and emphasizes on both technical and managerial skills. Software Engineering is a practice-oriented and applied discipline and hence there is an emphasis on hands-on development, process, usage of tools in addition to theory and basic concepts. We present an approach for mining the process data (process mining) from software repositories archiving data generated as a result of constructing software by student teams in an educational setting. We present an application of mining three software repositories: team wiki (used during requirement engineering), version control system (development and maintenance) and issue tracking system (corrective and adaptive maintenance) in the context of an undergraduate Software Engineering course. We propose visualizations, metrics and algorithms to provide an insight into practices and procedures followed during various phases of a software development life-cycle. The proposed visualizations and metrics (learning analytics) provide a multi-faceted view to the instructor serving as a feedback tool on development process and quality by students. We mine the event logs produced by software repositories and derive insights such as degree of individual contributions in a team, quality of commit messages, intensity and consistency of commit activities, bug fixing process trend and quality, component and developer entropy, process compliance and verification. We present our empirical analysis on a software repository dataset consisting of 19 teams of 5 members each and discuss challenges, limitations and recommendations.

Acknowledgments

I would like to express my sincere gratitude to my advisor Prof. Ashish Sureka for his continuous support in my M.Tech study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. Without his guidance and persistent help this thesis would not have been possible.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Sanjay Goel and Prof. Rahul Purandare, for their encouragement and support.

I would also like to thank my fellow mates Ritika Jain and Swati Aggarwal for their encouragement, insightful comments and suggestions.

In addition, I would like to thank my family for supporting me throughout my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Research Motivation and Aim

Software Engineering (SE) is a practice-oriented and applied discipline and software development processes, team-work, project management and exposure to popular SE tools are important learning objectives in addition to fundamental technical concepts in SE courses [2] [5] [6] [9] [10] [12] [15] [16]. SE courses generally consists of a team-based semester-long project giving an opportunity for students to apply the theory and principles learnt during lectures. Student teams follow a development process to deliver a software product. Several artifacts such as software requirement specification document (SRS, design document, project plans, test plans and source code) are produced at various milestones during product development [2] [5] [6] [9] [10] [12] [15] [16]. A course instructor can easily assess and provide feedback on product and deliverables such as SRS as it is visible but providing feedback on process and team-work is not straightforward as it is not explicitly visible unlike a product. The study presented in this paper is motivated by the need to provide an effective mechanism for SE course instructors to gain visibility and insights on software development processes followed by student teams and provide appropriate feedback on process improvement. Our motivation is to develop tools and techniques for solving problems encountered by SE course instructors.

Several SE tools such as online Wiki based collaborative document editor, Version Control Systems (VCS), Issue Tracking System (ITS), project management and testing tools facilitating team-work and product development are taught during the course. Tools such as ITS, VCS and Wiki generate an event log of activities performed by developers. For example, VCS records who, when and what of a software change. Wiki event log records who (actor) made what change and when (timestamp). Event logs generated by such Process Aware Information Systems (PAIS) can be process mined for discovering run-time process map, imperfections, inefficiencies and interesting patterns [16]. Process mining event logs generated from PAIS used by student-teams in an education setting can provide feedback and actionable information to the course instructor. The specific research aim of the work presented in this paper is the following:

1. To investigate the application of process mining event log data generated from Wiki based

Study	University	Repository	Objective
Glassy 2006 [2]	University of Montana	SVN	Mining problematic patterns, steadiness of progress and quality of commit messages
Jones 2010 [5]	Bloomsburg University	SVN	Aid in determining the accomplishments of each individual in a group programming project.
Kay 2006 [6]	University of Sydney	Wiki, ITS and SVN	Frequent pattern characterizing aspects of team-work.
Liu 2004 [9]	University of Alberta	CVS	Studying correlation between grades and nature of collaboration.
Mierle 2005 [10]	University of Toronto	CVS	Mining statistical patterns or predictors of performance.
Poncin 2011 [12]	Eindhoven University of Technology	SVN, Mails, Wiki Articles Logs	Identifying developer roles, development models and prototype reuse
Robles 2013 [15]	Universidad Rey Juan Carlos	GIT VCS	Gathering software analytics data from programming assignments

Table 1.1: Seven Closely Related Work Arranged in Chronological Order and Categorized Based on Three Attributes

collaborative document editor, Version Control Systems (VCS) and Issue Tracking System (ITS) in the context of an undergraduate SE course.

2. To define new process quality metrics and visualization and examine the application of existing process quality metrics for the purpose of understanding software development process by student teams in an educational setting

3. To conduct a case-study and empirical analysis of process mining software repositories and activity logs in a semester-long undergraduate level SE course and present our (teaching team) experiences, effectiveness of the proposed approach as well as challenges encountered.

1.2 Related Work and Research Contributions

In this Section, we present closely related work to the study presented in this paper and present novel research contributions in context to existing work. We characterize closely related work as formal studies on mining software repositories of student team-based projects within the context of university level Software Engineering course.

Table 1.1 presents the outcome of our literature survey on closely related work. We categorize 7 chronologically sorted papers (refer to Table 1.1) based on three attributes: software repositories used in the study, name of the University and objective of the research study. Our literature survey reveals that mining software repositories in an educational setting is an area that has attracted the attention of several instructors teaching university level SE courses and the study presented in this paper aims to further body the knowledge on the topic. Table 1.1 shows that mining VCS is the most common application and mining other software repositories like Wiki

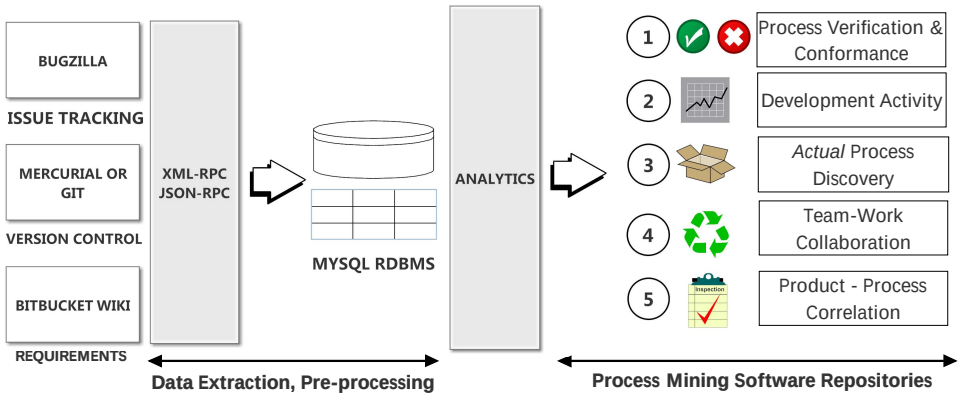

Figure 1.1: Research Framework

Field	Value	Field	Value
Total Students	99	VCS : Total Commits	925
Total Teams	19	VCS : Average Commits	50
Date of First Wiki Activity	21-08-2013	Date of First ITS Activity	04-09-2013
Date of Last Wiki Activity	10-10-2013	Date of Last ITS Activity	17-10-2013
Wiki : Total Commits	1167	ITS : Total Bug Reports	482
Wiki : Average Commits	60	ITS : Average Bugs Reported	26
Date of First VCS Activity	23-08-2013	ITS : Maximum Bugs Resolved	404
Date of Last VCS Activity	17-10-2013	ITS : Average Bugs Resolved	22

Table 1.2: Experimental Dataset Details

based content management system and issue tracking system is relatively unexplored. In context to existing work, the novel research contributions of this paper are following

1. We present an application of process mining Wiki, VCS and ITS event logs in a SE course setting consisting of 19 teams of 5 members each. While there are several studies on mining VCS, analyzing wiki and ITS activity event logs from a process mining perspective is a fresh perspective presented in this paper.

2. We present several visualizations and metrics providing feedback on various process aspects such as: work-load distribution between team members, regularity in contributions from start till the deadline, quality of commit messages, component and developer entropy, quality of efficiency of bug fixing process.

3. We conduct a case-study and apply the proposed approach on software repository dataset generated from 19 student projects in an undergraduate level SE course. We present an experience report, results and challenges encountered.

1.3 Research Methodology and Experimental Dataset

We propose a research framework (called as Mimansa) illustrated in Figure 1.1 and conduct experiments on dataset described in Table 1.2. As shown in Figure 1.1, the architecture of Mimansa consists of multiple components and phases. We mine three process aware information systems (PAIS): Bugzilla[1] issue tracking system, Mercurial[2] and Git[3] version control system and Bitbucket[4] Wiki. We extract raw data from PAISs using XM-RPC and JSON-RPC APIs calls and store the data in a MySQL database. The extracted data is then pre-processed and transformed to a format which is suitable to a mining task (for a given objective). Fields which are common across event logs from all the three systems are: activity timestamp, actor and the action. We frame several questions (actionable information for the Instructor such as the degree of contribution from team members towards a common goal) and define process quality metrics and visualizations which are addressed in the Analytics phase of the research framework. As shown in Figure 1.1, the outcome of process mining is multi-dimensional and derived by mining data from multiple PAIS giving visibility to the instructor on the development process in addition to the product.

Software Engineering (CSE 300) is a 4 credit core course (two classes of 1.5 hours each in one week with a total of 26 classes over 4 months) offered during the third year (5^{th} semester) of Bachelor of Technology (abbreviated as B.Tech) in Computer Science and Engineering (CSE) program at Indraprastha Institute of Information Technology (IIIT Delhi, a state university in India). The experience report is based on SE course taught in Monsoon 2013 semester. We created 19 teams of 5 members each (except few teams which had 6 members). Table 1.2 describes the experimental dataset collected over a period of 8 weeks. The start date for the experimental dataset is the begin date of wiki activity (21-August-2013) until the last date of ITS activity (17-October-2013). As shown in the Table 1.2, we observe a total of 1167 commits in Wiki, 925 commits in VCS and 482 issues in Bugzilla. We notice that 404 out of 482 issues reported to ITS are resolved. We defined several project deliverables during the course. The first project deliverable was on creating $25-30$ user stories and a release plan (using team wiki). One of the project deliverables was to conduct functional and non-functional testing of the software and use ITS for managing and tracking defect reports as well as feature enhancement requests.

[1] http://www.bugzilla.org/
[2] http://mercurial.selenic.com/
[3] http://git-scm.com/
[4] https://bitbucket.org/

Chapter 2

Team Wiki

We taught basic principles and practices of Extreme Programming (XP) during the course and one of the project deliverables was on writing user stories, product backlog and release plan according to Extreme Programming methodology [11]. Agile development processes are intended to support early and quick production of working code by structuring the development into small release cycles and focussing on continual interaction between developers and customers. User stories are one of the primary development artifact of this approach and are used as the basis for defining the functions a system must provide and to facilitate requirements management [13]. It captures the 'who', 'what' and 'why' of a requirement in a simple, concise way specifying the needs and expectations of the user. Team Wiki's were used by the students for the documentation of user stories. A wiki is a collaborative tool that supports software development and maintainence process of a team project. It is designed to facilitate the exchange of information within and between teams.

Whatever changes are done to the information in a wiki leaves its traces in the event log produced by wiki that can be reviewed later to retrack the process. A wiki event log represents the activity of developers during requirement specification documentations. Figure 2.1 is a snapshot of Bitbucket Wiki feature in which we mask the actual names of the authors and label various data fields. From this log, we obtain information like a timestamp corresponding to an activity, summary of commit and its author. Using this information we analyze the process and practise followed by students during the software development.

2.1 Quality of Commit Messages

Wiki commit messages are like documentation and writing quality and well-formed commit message is a process attribute and not a product (a document containing user-stories) attribute. We emphasized in class on writing good quality commit messages (making a habit of writing high quality commit messages) as it results in good documentation, gives a better insight and helps in debugging or answering questions. We conducted a qualitative analysis of the commit

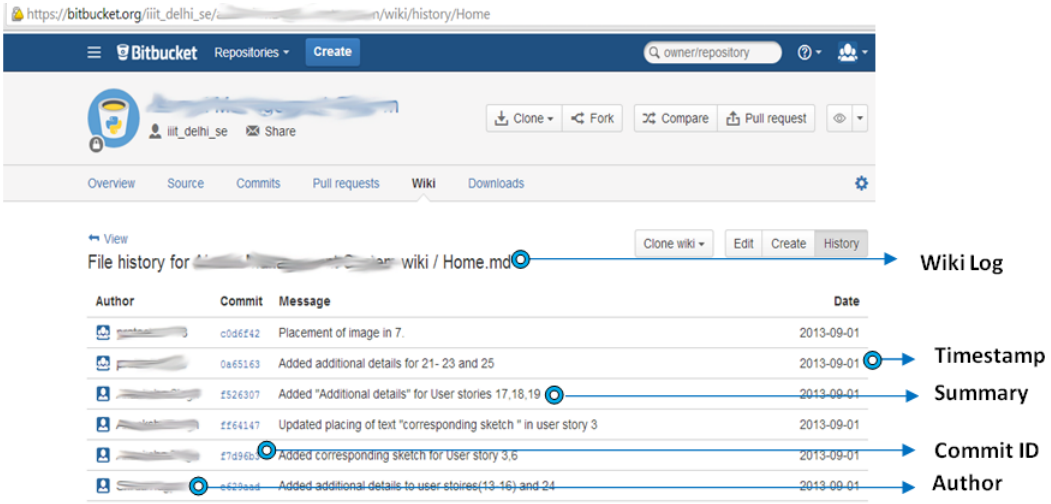

Figure 2.1: A Snapshot of Wiki Event Log

messages to check if they are clear and concise, explains the rationale and mentions the concepts like how and why this change was committed. Following is the process quality question useful to the instructor from pedagogy perspective:

RQ1 : *To what extent guidelines on commit message quality are followed and what are the specific gaps.*

We categorize the quality of commit messages into three broad classes: Good, Average and Poor. Table 2.1 shows examples of few commit messages in each category from student projects. We observe commit messages like “Edited some user stories” in which it is not clear which user story is edited. We notice commit messages like ”Edited online”, ”Minor fixes here and there” and ”Wrote 4 user stories” which are vague and poor quality. We also observe several commit messages (refer to Table 2.1) like ”Added title, priority and cost for user stories 17 and 19” which is short, concise and specific. Quality of commit messages was a parameter for grading the project deliverable and the grade was not only based on product quality but also process quality.

2.2 Consistency in Commit Activity

Software Engineering is a 15 weeks long course with a project that should evolve in an incremental and iterative fashion(reflecting a real world development environment). Analysing the commit activity of a team wiki helps in understanding the nature of evolution of a project during the documentation of user stories. We define the commit activity of a team as the frequency of commits across timeline. A consistent pattern in the frequency of commits by a team reflects an efficient pattern in the evolution of a project. Analyzing the frequency in commit activity will help the instructor in answering the following research question:

RQ2 : *To what extent the process followed by the students during the requirement documentation*

Quality	Example	Remarks
Poor	1. Edited online. 2. Wrote 4 user stories. 3. Minor fixes here and there. 4. User Story-32	Vague, not specific, only 2-3 terms, too general.
Average	1. Changes to story 8,9,10,14. 2. Edited some user stories(Priority heading removed). 3. Edited 2nd user story. 4. Image issues of 20.	Specific but still not covering all (rationale, why and what) relevant details.
Good	1. Re added costs for 9-12,got removed due to commit conflict. 2. FAQ page addition for easier usage of the application. 3. Added Title, Priority and Cost for User Stories 17,19. 4. Release of user story 1 and 2 added.	Short, concise, specific and explains the rationale, why and what.

Table 2.1: Categorization of Commit Messages into Three Levels of Quality

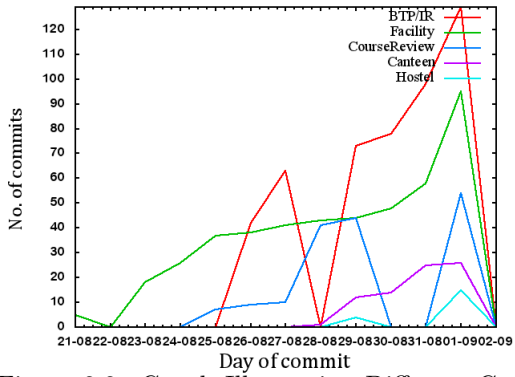

Figure 2.2: Graph Illustrating Different Commit Activity (Number of Commits Across Time) Patterns

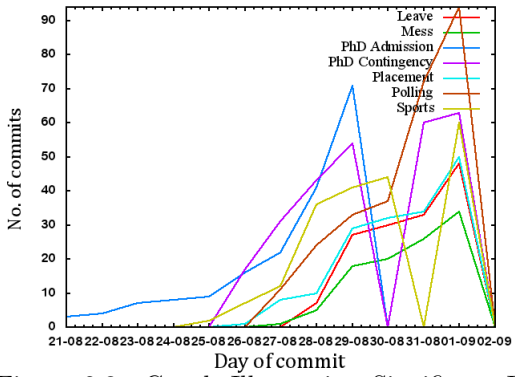

Figure 2.3: Graph Illustrating Significant Increase in Number of Commits Close to the Deadline

of their projects is incremental and iterative.

Figure 2.2 shows different commit activity patterns of the teams. We notice an incremental pattern in the frequency of commit activity for team like "Facility" and at the same time a non-incremental pattern can be noticed for team like "Hostel". Reviewing the commit activity of teams across timeline unfolds another important aspect of an educational environment and that is reviewing the amount of work done by students near to the deadline. From Figure 2.3 we observe that most of the teams follow a non-uniform process and work more frequently near to the deadline. Figure 2.3 reveals that team "Polling" contributes 35% of the total work on the last day of the deadline. While we see an increase in activity in last 2 days, there is work in the start and middle as well, as can be seen in projects "PhDAdmission" and "PhDContingency". Project work for team "PhDAdmission" started early and finished off well before the deadline. Figure 2.2 and Figure 2.3 shows multiple patterns observed from wiki activities of different

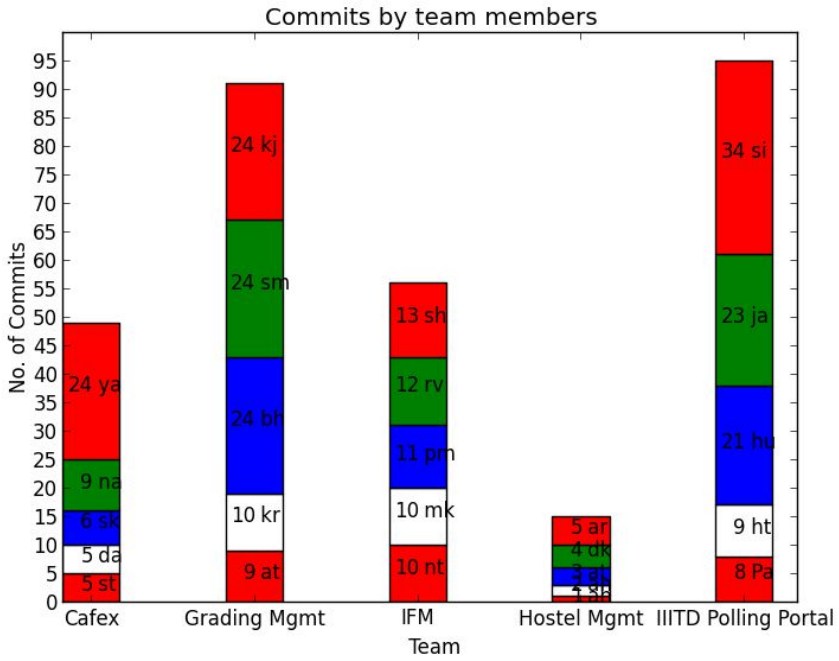

Figure 2.4: Contribution of Each Member of a Team in a Wiki

teams, though we do not plot patterns for all the projects and represent the activity of only a few projects.

2.3 Contribution of Members in a Team Wiki

Software Development Process inherently requires team work with an equal distribution of work load across all the members of a team for the development of a healthy product. We encouraged students to put an emphasis on uniform team work with an aim to develop the quality of team spirit amongst them. Student team projects are a hallmark of undergraduate Software Engineering courses and programs [4] [14]. We look into the contribution of team members in a team wiki to provide an insight to the instructor from the following point of view:

RQ3 : *To what extent the work load distribution in a team was uniform or skewed.*

A stacked chart is used as shown in Figure 2.4 to represent the distribution of work across the members of teams. Figure 2.4 shows different levels of work load distribution across different members of teams. We observe that team members can be divided into 3 broad categories namely: Out-performers as shown by team "Cafex" with member "ya" contibuting 45-50% of the total work, Equal performers, team "IFM" is an example of this category in which each member contributes 20-25% each and below expectation performers like in team "IIITD Polling Portal" where member "Pa" is contributing only 8% of the total work. Such a differentiation in students further benefits the instructor in adopting a fair grading policy.

Chapter 3

Version Control System

Version Control Systems(VCS) is an essential tool in software developement of a project. In an educational setting, use of VCS is not only beneficial for students as it prepares them for the real life situations but has several potential benefits for the instructor as well. It can be used as a tool to monitor or visualize team and individual contribution. It helps in reviewing the development process followed by the student teams across a timeline and their behiovior near the release dates.

The event log of a VCS is the most important documentation for any project. We analyze the event log of the projects to get an insight on process followed by student teams during the development phase.

3.1 Component And Developer Entropy

SE courses generally includes a team-based semester-long project where student teams follow a development process to deliver a software product. Several artifacts such as software requirement specification document (SRS), design document, project plans, test plans and source code are produced at various milestones during product development. A course instructor can easily assess and provide feedback on product and deliverables as such as SRS as it is visible but providing feedback on process and team-work is not straightforward as it is not explicitly visible unlike a product. In student projects, analysing the contribution of individual developers in a team not only helps the instructor in understanding the quality of team work involved but also helps in fair grading process. Every project can be divided into several modules or components which require a collaborative effort from all members of a team to yield an efficient product. The students were asked to follow a modular structure for the development of their project. We study the uniformity in the contribution of developers across all the components of a project to facilitate instructor understanding the following:

RQ4 : *To what extent the contribution of work amongst the developers of a team across various components was uniform.*

We define that team work in a project will be perfect if the contribution of all the members of the team is uniformly distributed across various components of the developed software. That is, if a team shows equal work from all the members in a component and across all the components of the software then it shows perfect team work. We examine the VCS event logs to determine the contribution of each developer across all the components of a project. A Radar Chart as shown in Figure 3.1 is used to represent the contribution of all the team members across various components. Each axis represents a component of the software and each colour represents a developer. An equal distribution of a colour along all axes represents equal work by a developer. A skewed distribution of a colour shows an expertise area of a developer but shows lack of participation in all the components of the software.

Figure 3.1 shows the contribution of developers of a team within a component and across the components. It shows participation of member2 equally spread to all the components whereas a peak in the contribution of member3 and member4 in components "Authentication" and "Systems" respectively shows their expertise in the respective areas but lack of contribution to other parts of the software. It also helps in spotting the least contributors of a project as well like "member1" who worked least in the entire team.

We define metrics called as Component Entropy (refer to Equation 3.1) and Developer Entropy (refer to Equation 3.2) to quantify two types of distribution (developer-component and component-developer) serving as an indicator of extent of co-development at component level. Lal et al. [8] apply the concept of Entropy for calculating component and bug-reporter entropy across seven different types of defect reports and Khomh et al. [7] present an application of bug triaging by computing crash-entropy (calculating entropy based on crash-types across users).

$$H(d|c) = -\sum_{j=1}^{j=m}\sum_{i=1}^{i=n} p_{(d=i|c=j)} * log_n(p_{(d=i|c=j)}) \tag{3.1}$$

$$H(c|d) = -\sum_{i=1}^{i=n}\sum_{j=1}^{j=m} p_{(c=j|d=i)} * log_m(p_{(c=j|d=i)}) \tag{3.2}$$

We instructed the student teams to have a modular design of their system consisting of components or modules. A component consists of a collection of files and components are interdependent. If two developers modify same files or files within the same component then we refer it to as sharing, co-development or collaboration at the component level.

Let us say a team of n members is developing a system consisting of m components. Developer-component entropy (Equation 3.2) quantifies the distribution of n developers across m components. Inner summation calculates the individual entropy of each developer which is then averaged over all the developers. Similarly, Component-developer entropy (Equation 3.1) refers to the distribution of each of the m component across n developers. Here, Inner Summation represents the entropy value of individual components where a high value indicates an equal contribution from all the developers in this component. The entropy value can vary from a minimal of 0 to a maximum of 1. The entropy is maximal (one extreme end) if there is a perfectly uniform

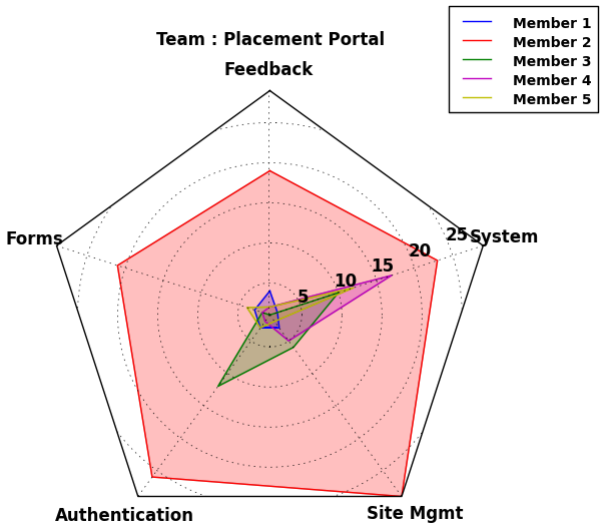

Figure 3.1: Radar Chart Illustrating Developers Contribution Across Components

distribution (every component receives equal contribution by all the team members). On the other extreme end, entropy value is minimal if there is no co-development or collaboration at component-level.

Figure 3.1 displays the level of collaboration in the developement of project "Placement". The developer entropy for member2 is 0.99 showing his equal contribution to all the components. The overall Developer-component entropy for "Placement" is 0.87. Similarly, we calculate the component entropy for component "Feedback" as 0.64 reflecting unequal contribution of all the team members in this part of the project, on the other hand the component entropy for component "System"is 0.88. The overall Component-developer entropy is coming out to be 0.72. Figure 3.2 represents the component and developer entropy and the frequency of commits for 6 teams. We notice that for team "Leave", component and developer entropy value lie in a low value region of 0-0.5 representing poor team work whereas high entropy value for teams like "Grading", "Alumni", "Placement" and "Mess" indicate collaborative development of the projects. The entropy values obtained from the above mentioned metrics reflects the diversity in the contribution of work in a project and not the intensity. One of the limitations of this metric is that if a developer works less but still shows uniformity in his contribution to all the components, then we get a high value of developer entropy. The same applies to the component entropy as well. One way to extend this metric and remove this limitation is to add weights or a bias during the calculation of the entropy values.

3.2 Effects of Milestones on Commit Behaviour

The development process of a software is an incremental process that can take multiple iterations offering a series of releases with additive functionality. Analyzing the development activity of projects near release dates helps in understanding the change in the behaviour of students with deadlines. We focussed on the commits that involve source code contributions of different developers of a project and their commit frequency to understand their behaviour near pre-set milestones or release plans to help instructor understand the following aspect:

RQ5 : *To what extent did various milestones or release plans effected the activity of developers*

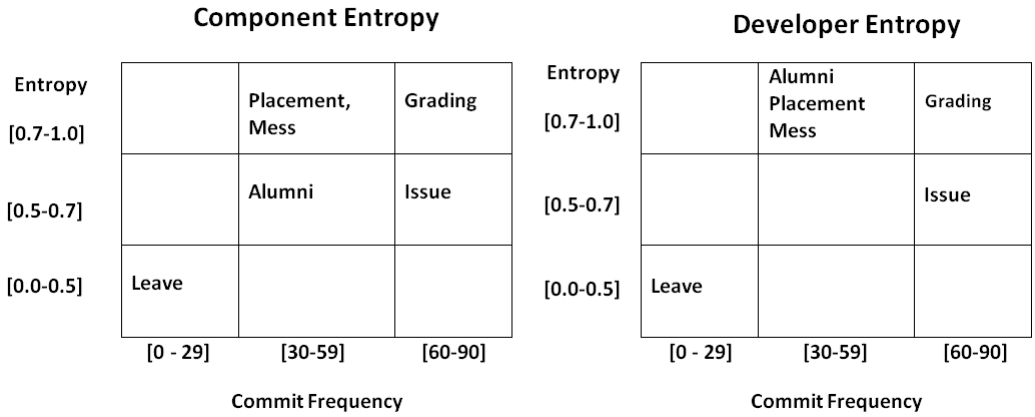

Figure 3.2: Component and Developer Entropy Graphs Showing the Position of 6 Teams

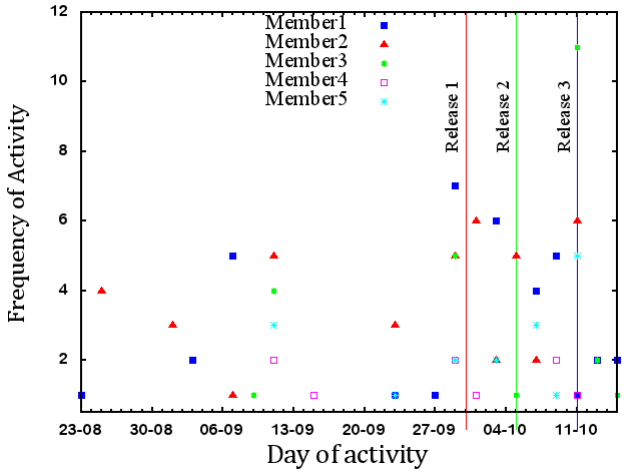

Figure 3.3: Scatter Plot Representing the Commit Activity with Release Dates

of a project.

Figure 3.3 shows the commit frequency of different developers of team "CourseReview". Each colour represents a different developer and its position in the graph represents the amount of work done in terms of number of commits. Release dates associated with this project are represented by the vertical lines. An increase in commit frequency shows an increase in the development activity of a project. Figure3.3 represents an increase in the density of commit activity near the release plans with an increase in number of commits by majority of the developers. From this we infer that developers become more active near to the release of a deliverable and their work is not uniformly distributed throughout the development process. We also notice that member2 shows a consistent behavior with decent commit activity throughout the timeline indicating a member of the team who takes timely actions in the development process of the project.

Chapter 4

Issue Tracking System

Issue Tracking System such as Bugzilla are Process Aware Information Systems used during software development and maintenance for issue (feature enhancement requests and defects) tracking and management of a project. Software Maintenance and defect-fixing process is a collaborative activity involving several roles such as bug reporter, bug fixer and project developers. 4.1 shows the main page of Bugzilla which is accessed to search for reported bugs or to report new bugs. An ITS produces an event log or the history of the events that were followed during bug-fixing process. 4.2 represents event log of a bug with information like who reported a bug and when and changes in resolution and status of a bug with their respective timestamps.

Process data archived in an ITS as an event log can be mined to track the bug-resolving process of a project and can also be used for contribution and performance assesment of individuals as a bug reporter and fixer. In the Software Engineering course, Bugzilla was introduced as an ITS to be used for the defect tracking and maintenance of their products. Each team project was allocated a testing team that was responsible for testing their projects. Team members were also responsible for testing their own projects apart from the testing of projects allocated to them. In the following sections we provide visualizations and metrics to accurately and reliably measure the contribution of students teams as developers and testers during the software maintenance and bug-fixing process.

4.1 Bugs Opening Trend, Closing Trend and Continuity

Francalanci et al. [1] present a method to measure performance characteristics such as continuity and efficiency of bug fixing process during software maintenance [1]. They define two performance indicators (bug opening and closing trend) reflecting the characteristics and quality of bug fixing process. We mine the issue tracking system of student projects and apply the performance indicators proposed by Francalanci et al. to measure the bug fixing performance efficiency [1]. Francalanci et al. define bug opening trend as the cumulated number of opened bugs over time and closing trends as the cumulated number of bugs that are resolved and closed

Figure 4.1: Bugzilla Main Page

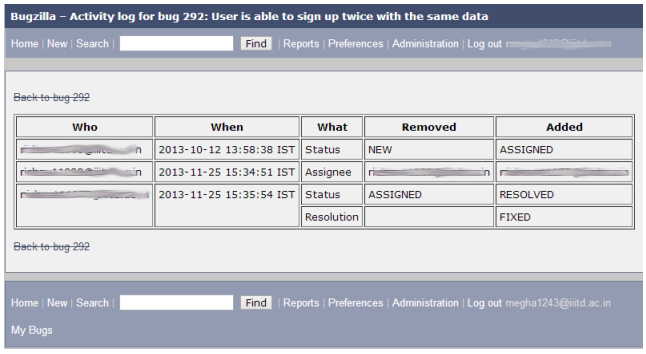

Figure 4.2: A Snapshot of Bug History

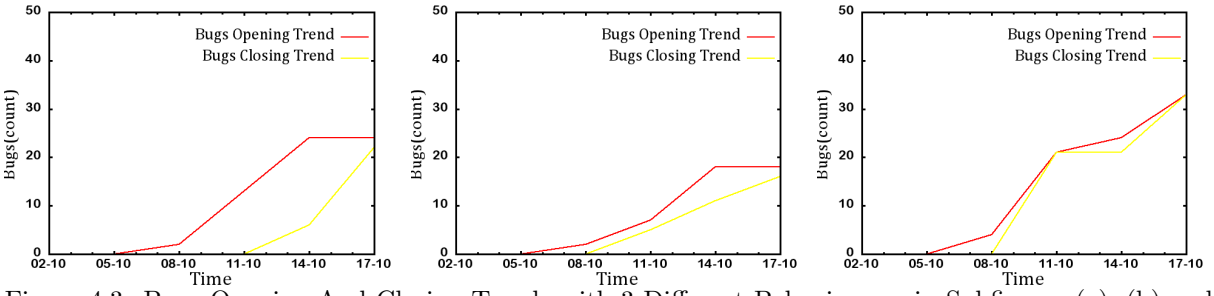

Figure 4.3: Bugs Opening And Closing Trends with 3 Different Behaviour as in Subfigures (a), (b) and (c)

over time [1].

We plot the opening and closing trend for all projects on a graph and investigate the similarities and differences in their characteristics. In Figure 4.3, x-axis represents the timeline across which bugs are reported and resolved and y-axis shows the bugs count associated with each date. Figure 4.3 displays the opening and closing trend for three projects exhibiting different characteristics. At any instant of time, the difference between the two curves (interval) can be computed to identify the number of bugs which are open at that instant of time. We notice that the debugging process is of high quality for the graph in Figure 4.3(c) as there is no uncontrolled growth of unresolved bugs (the curve for the closing trend grows nearly as fast or has the same slope as the curve for the opening trend) across all time instants. However, we notice in Figure 4.3(a) that the growth of unresolved bugs increase with time and are resolved only when the deadline approaches. This shows an inefficient growth and process of software testing. Figure 4.3(b) shows a team with an average trend of bug fixing process.

We propose a metric called as "Bug Fixing Score" as shown in Equation 4.1 to quanitfy the bugs opening and closing trend of a project. The metric can serve as an indicator of the bug fixing performance efficiency.

$$BugFixingScore = (1/n - i) * \sum_{i}^{i=n} log(BO_i - BC_i)/log(BO_i) \tag{4.1}$$

where, BO = Bugs opened till time instant i , BC = Bugs closed till time instant i , "i" is the first time instant when BO¿0 and "n" is the maximum recorded time instant. Score of a project

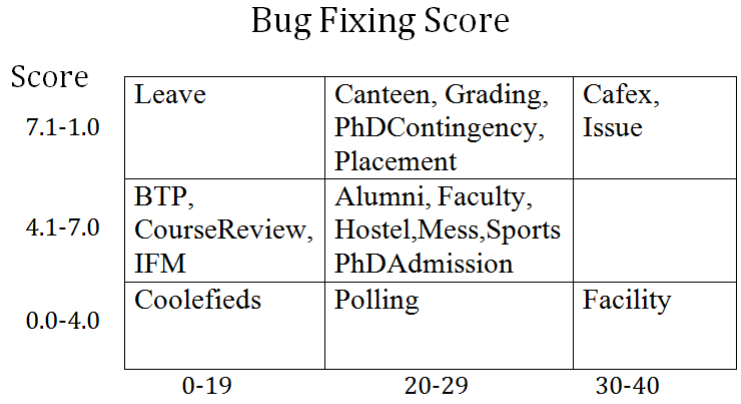

Figure 4.4: Bugs Fixing Score Graph Showing the Position of 19 Teams

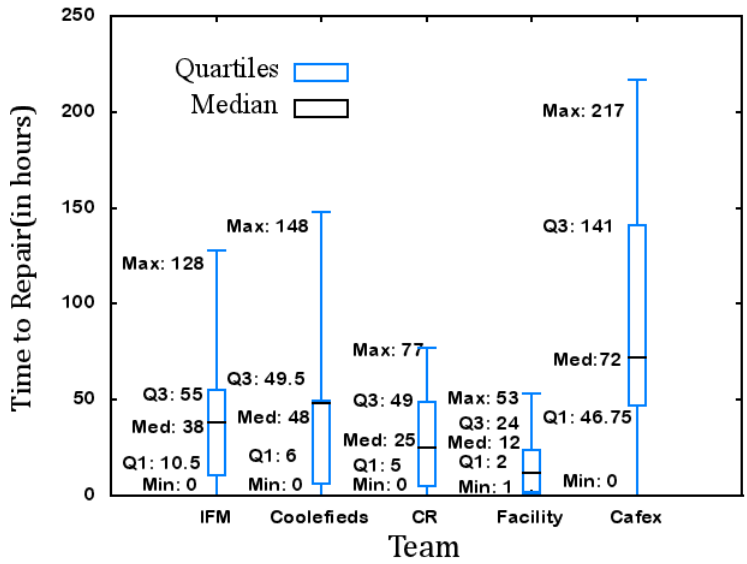

Figure 4.5: A Box Plot Illustrating the Statistics for Time to Repair of Bugs Reported

can vary from a minimal value of 0 to a maximal value of 1 where value 0 indicates a perfectly efficient behaviour where the bug fixing process is as fast as the bug opening process. On the other extreme end, score value 1 indicates an inefficient behaviour where there is no bug fixing throughout the process though number of bugs opened continue to increase over time. The Bug Fixing Score helps the instructor in answering the following research question:

RQ6 : *To what extent the bug fixing performance of a project was efficient.*

We calculate the score of all projects as shown in Figure 4.4 and place them according to the value of their score with respect to number of bugs reported. In Figure 4.4, x-axis represents the count of bugs reported and y-axis shows bug fixing score divided over 3 intervals . We conclude that 50% of the projects exhibited a normal behaviour in bug-fixing process with a score value in range 4.1-7.0 whereas only 16% of the teams showed an efficient behaviour in the bug-fixing process lying in a range between 0.0-4.0. Rest of the teams showed an inefficient behavior in

the bug fixing process. In Figure 4.4, bottom right corner represents a team which reported considerable number of bugs with an efficient value of bug fixing score lying in the range of 0.0-4.0. From this graph, one can easily comprehend and visualize the performance of various teams during the testing and software maintenance phase of projects. The instructor can take benefit of this metric during grading of projects by understanding the efficiency of student teams during the bug reporting and resolving process of their respective projects.

4.2 Mean Time to Repair

Time taken to repair a bug (TTR) is a measure that helps in determining the efficiency of the developers during the testing phase of a project. We define the time to repair a bug as follows:

$$TTR(b)(inHours) = date_{closed}b - date_{opened}b \tag{4.2}$$

Analysing the repairing activity obtained from the time taken by developers to resolve bugs helps in quantifying the devotion and importance given by the development team to the testing and maintenance process of their projects and serves as a measure of the seriousness of developers to the issues faced by their customers. The statistics obtained from bugs repairing time of student teams helps the instructor to comment on the following research question:

RQ7 : *What are the variations in the time devoted and importance given by the development teams to the repairing activity of bugs.*

We use a box plot as shown in Figure 4.5 to represent the statistics related to the TTR of bugs for different teams. Figure 4.5 shows the quartiles and median values for 5 teams. A high value of "Q3" indicates that bugs in this project were left unnoticed or no importance were given to the bugs resolving process thus shedding light on the irresponsible behaviour of the development team towards importance of testing process. From Figure 4.5 high value of Q3 for team "cafex" indicates an inefficient behaviour of developers of this team towards the bug resolving process. On the other hand we observe that most of the teams are showing an efficient behaviour in bug resolving process taking time between 5 to 60 hours ,that is, the bugs are getting resolved within a timespan of 2 days from their reported time. Resolving bugs in sufficient time not only reflects the sincerity of the development team towards the maintenance process of their product but also motivates them to avoid such situations in their future releases.

4.3 Component Vs Priority in a Project

A software project can be divided into various sub-modules or components developed by the collaborated efforts of the developers of a project. Rigorous testing and inspection is required in each part of a project for successful and reliable delivery of the product. During the testing phase of a software, all the components are tested and bugs with different priority are reported

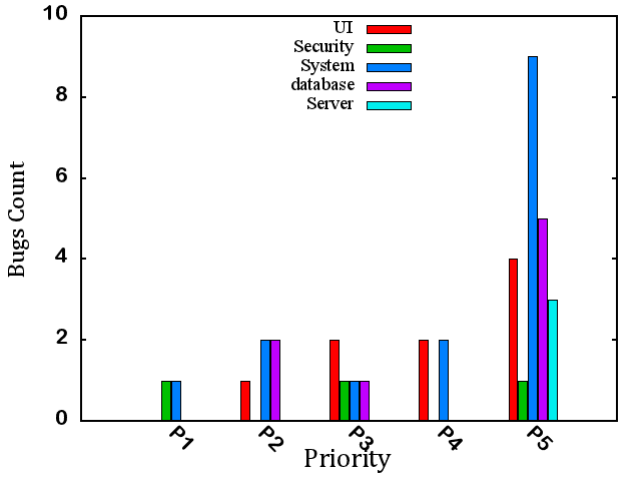

Figure 4.6: A Histogram Showing the Distribution of Bugs Priority Across Various Components of a Project

according to the importance of their functionality. A component with majority of the bugs in high priority region not only indicates its importance from the users point of view and urgency to correct them but also shows an unsuccessful attempt of the development team in serving the main functionality of this component. For the student projects, such a review will help the instructor in identifying:

RQ8 : *What are the components of a project that turned out to be weak and less functional during the testing phase and what are the reasons for such a behavior.*

Figure 4.6 is a histogram which shows the distribution of bugs with different priority in various components of a project. We notice that maximum number of bugs are reported in component "system" that shows its importance in the project and indicates lack of expertise in serving the main functionality of this product. On the other hand, component like "security" with less number of bugs shows the expertise and successful attempt of the project team in serving the need of the user. In an educational setting, such a graph also helps the instructor during the viva of the student projects as the instructor can ask the reasons for the failure of a particular component, who was responsible for the development of this part and what steps were taken to avoid such situations in the future releases of the project.

4.4 Process Discovery

Event log archived in ITS stores information like what was changed as in status or the resolution of a bug and the timestamp associated with the change. We mined the event logs collected from the bugs activity of student's projects and mined it to get an insight into the process followed by student teams during the software testing and maintenance phase. In the Software Engineering course, the instructor emphasised student teams to follow design time process model of a bug lifecycle during bug fixing process. We review the process models of the bugs activity in projects

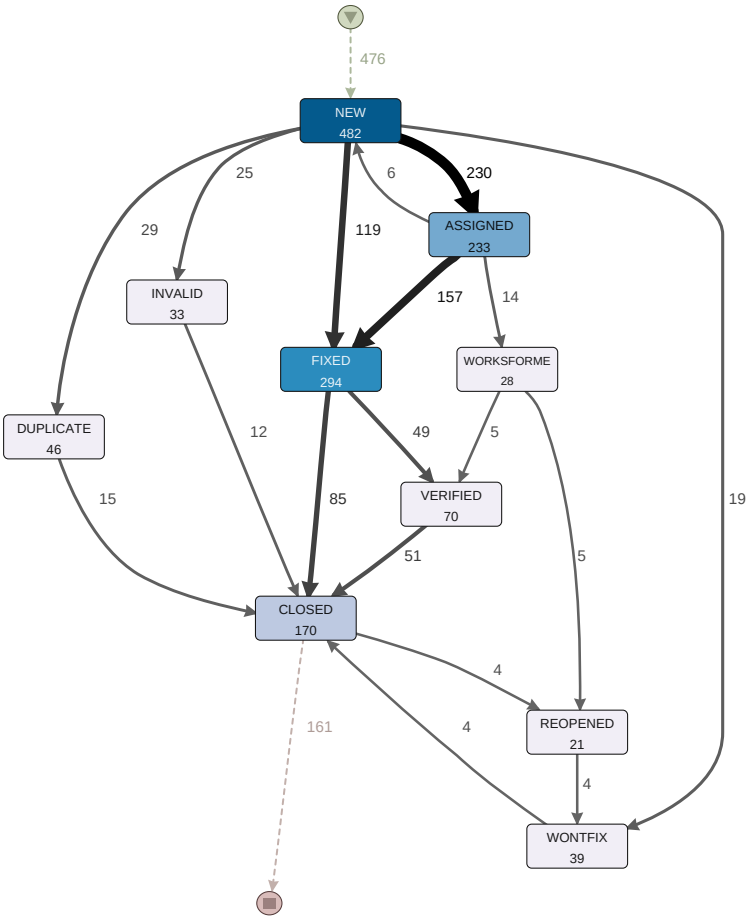

Figure 4.7: Run-Time Bug Life-Cycle Process Map

to give an insight to the instructor from the following perspective:

RQ9 : *To what extent the runtime process models of student teams conform with design time process model.*

There are many process mining tools like ProM(Open source) and Disco(commercial)[1] used to obtain process models of softwares. We used Disco[2] as a tool to obtain process map and other statistical information for testing process of student projects. Disco miner is based on the proven framework of the Fuzzy Miner with completely new set of process metrics and modelling strategies[3].Preprocessed data is imported into Disco which is used to discover the runtime or actual process from the event log generated during the progression of a bug. We provide the data collected from event logs to Disco with 3 types of information: Bug ID as the Case ID, Status field or resolution field as activity, and event log timestamp as timestamp. Figure 4.7 reveals the bugs activity process of the student projects. Each node represents an activity and an edge between two nodes represents a transition from one activity to another during the bug

[1]Disco is a proprietary tool for which we availed academic license and used for our analysis.
[2]http:// fluxicon.com/disco/
[3]http:// fluxicon.com/disco/files/Disco-Tour.pdf

	Alumni Mgmt	**CourseReview**
Fitness Metric	0.384	0.793
Total unique Transitions	72	72
Total Inconsistent Transitions	22	6
Most Inconsistent Transition	$Resolved \rightarrow Closed$	$New \rightarrow Resolved$
Frequency of Most Inconsistent Transition	15	6

Table 4.1: Fitness Evaluation and Compliance Verification

lifecycle. Total reported bugs are 476 which travel through 10 different stages in their lifecycle represented by 10 nodes. Label of each edge indicates the absolute frequency of transition, shade and thickness corresponds to the frequency with more frequent being dark and less frequent as light. We notice that 29% of bugs were marked as "closed" right after they were "fixed" without getting "verified". Around 7% and 5% of the bugs were reported as "invalid" and "wontfix" respectively. Process Map shown in Figure 4.7 was further analysed to review activity frequency and transition frequency . We observe that once a bug is reported as "new" the most frequent event that it passes through is "assigned" followed by "fixed". We infer from the analysed process map that $New \rightarrow Assigned$, $Assigned \rightarrow Fixed$ and $New \rightarrow Fixed$ are the most frequent transitions.

4.5 Compliance Verification

[3] presents a method to detect the inconsistencies between design time process model and the model obtained for as-is process from runtime event log. [3] defines a metric to measure fitness, that is, how well observed process complies with the control flow defined in the design time process model and the point of inconsistency.

They propose an algorithm to evaluate fitness metric to find number of cases with valid traces (only defined transitions) and its ratio with total cases to measure the extent of fitness. Event log and adjacency matrix, A (with row as source state and column as destination state, that is, 6x6 in our case) has 1 in the cell if transition is preferred otherwise 0, are given as input. For example, $New \rightarrow Assigned$ is a recommended transition and hence the value in adjacency matrix A with row as "New" and column as "Assigned" will be 1, similarly on the other hand, $New \rightarrow Resolved$ is a permitted transition but not a recommended one and hence the value for such a transition will be 0 and such a transition will be considered as an inconsistent transition.

Using the proposed algorithm, an event trace (array with states from event log in sequential order of their occurrence) for each case ID as shown in step 4 of algorithm 1. For optimization, unique traces and count frequency of each is also identified. Each unique trace is verified with adjacency matrix A for conformance. If it has all permitted transitions then the valid bit V_i is assigned value 1 else 0. If the evaluated value of fitness metric is less than 1 then there is deviation from defined model. To detect the cause of inconsistency, [3] proposes another algorithm, that is, inconsistentDetector() which is invoked with event log and adjacency matrix,

A as input and returns inconsistency metrics. A footprint matrix is created of runtime event log and is compared with design model (adjacency matrix) to identify inconsistent transitions (non-zero elements in ITF). All states (6 for our case) are stored in an array, state. The frequency of transition between each pair of states is counted from event log and stored in Transition Frequency, TF matrix. Total inconsistent transitions are evaluated by adding the elements of ITF. The algorithm evaluates the frequency (maximum element of ITF) and most frequent inconsistent transition.

We implement the above mentioned 2 algorithms on the projects abtained from student teams to understand the inconsistencies obtained in the state transitions adopted by them in their bug resolving process. Analysing the fitness metric and performing conformance testing for student projects according to these algorithms, we obtain the results as presented in Table 4.1. Low value of Fitness Metric for "Alumni Mgmt" as compared to "CourseReview" indicates more deviation in the run-time process model of the former from the defined design time model. As shown in Table 4.1, for project "Alumni Mgmt", 30% of unique transitions are inconsistent transitions out of which most inconsistent transition is $Resolved \rightarrow Closed$ with frequency of 15 transitions from total of 22 inconsistent transitions. On the other hand, a better performance can be estimated for project "CourseReview" with most inconsistent transition as $New \rightarrow Resolved$ that accounts for only 8% of total unique transitions.

Chapter 5

Conclusions

Mining activity logs generated by process aware information systems such as Wiki based document management system, VCS and ITS as a result of developing software product in an education setting can present useful insights on run-time development process and process quality to the course instructor. We conduct a series of experiments which reveals that not only product but process quality varies significantly between student teams and mining process aspects can help the instructor in giving directed and specific feedback. We observe commit patterns characterizing equal and un-equal distribution of workload between team members, patterns indicating consistent activity in contrast to spike in activity just before the deadline, varying quality of commit messages, developer and component entropy, variation in degree of process compliance and bug fixing quality. We believe the proposed framework is effective in providing visibility to the instructor on process data which can be mined to derive actionable information for improving academic outcome and improve the teaching and learning methodology.

Bibliography

[1] Francalanci, C., and Merlo, F. Empirical analysis of the bug fixing process in open source projects. In *Open Source Development, Communities and Quality*. Springer, 2008, pp. 187–196.

[2] Glassy, L. Using version control to observe student software development processes. *J. Comput. Sci. Coll. 21*, 3 (Feb. 2006), 99–106.

[3] Gupta, M., and Sureka, A. Nirikshan: Mining bug report history for discovering process maps, inefciencies and inconsistencies, seventh india software engineering conference (ISEC).

[4] Hogan, J. M., and Thomas, R. Developing the software engineering team. In *Proceedings of the 7th Australasian conference on Computing education - Volume 42* (Darlinghurst, Australia, Australia, 2005), ACE '05, Australian Computer Society, Inc., pp. 203–210.

[5] Jones, C. Using subversion as an aid in evaluating individuals working on a group coding project. *J. Comput. Sci. Coll. 25*, 3 (Jan. 2010), 18–23.

[6] Kay, J., Maisonneuve, N., Yacef, K., and Zaïane, O. Mining patterns of events in students teamwork data. In *Educational Data Mining Workshop and Intelligent Tutoring Systems* (2006), pp. 1–8.

[7] Khomh, F., Chan, B., Zou, Y., and Hassan, A. E. An entropy evaluation approach for triaging field crashes: A case study of mozilla firefox. In *Proceedings of the 2011 18th Working Conference on Reverse Engineering* (Washington, DC, USA, 2011), WCRE '11, IEEE Computer Society, pp. 261–270.

[8] Lal, S., and Sureka, A. Comparison of seven bug report types: A case-study of google chrome browser project. In *APSEC* (2012), pp. 517–526.

[9] Liu, Y., Stroulia, E., Wong, K., and German, D. Using CVS historical information to understand how students develop software. In *MRS 2004: International Workshop on Mining Software Repositories* (2004).

[10] Mierle, K., Laven, K., Roweis, S., and Wilson, G. Mining student cvs repositories for performance indicators. In *Proceedings of the 2005 international workshop on Mining software repositories* (2005), MSR '05, pp. 1–5.

[11] Peeters, J. Agile security requirements engineering. In *Symposium on Requirements Engineering for Information Security* (2005).

[12] Poncin, W., Serebrenik, A., and van den Brand, M. Mining student capstone projects with frasr and prom. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion* (2011), SPLASH '11, pp. 87–96.

[13] Rees, M. J. A feasible user story tool for agile software development? In *Software Engineering Conference, 2002. Ninth Asia-Pacific* (2002), IEEE, pp. 22–30.

[14] Reichlmayr, T. Enhancing the student project team experience with blended learning techniques. In *Frontiers in Education, 2005. FIE '05. Proceedings 35th Annual Conference* (2005).

[15] Robles, G., and Gonzalez-Barahona, J. Mining student repositories to gain learning analytics. an experience report. In *Global Engineering Education Conference (EDUCON), 2013 IEEE* (2013), pp. 1249–1254.

[16] van der Aalst, W. M. P. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*, 1st ed. Springer Publishing Company, Incorporated, 2011.