Process Mining Multiple Repositories for Software Defect Resolution from Control and Organizational Perspective

Monika Gupta
IIIT Delhi
New Delhi, India
monikag@iiitd.ac.in

Ashish Sureka
IIIT Delhi
New Delhi, India
ashish@iiitd.ac.in

Srinivas Padmanabhuni
Infosys
Bengaluru, India
srinivas_p@infosys.com

ABSTRACT

Issue reporting and resolution is a software engineering process supported by tools such as Issue Tracking System (ITS), Peer Code Review (PCR) system and Version Control System (VCS). Several open source software projects such as Google Chromium and Android follow process in which a defect or feature enhancement request is reported to an issue tracker followed by source-code change or patch review and patch commit using a version control system. We present an application of process mining three software repositories (ITS, PCR and VCS) from control flow and organizational perspective for effective process management. ITS, PCR and VCS are not explicitly linked so we implement regular expression based heuristics to integrate data from three repositories for Google Chromium project. We define activities such as bug reporting, bug fixing, bug verification, patch submission, patch review, and source code commit and create an event log of the bug resolution process. The extracted event log contains audit trail data such as caseID, timestamp, activity name and performer. We discover runtime process model for bug resolution process spanning three repositories using process mining tool, Disco, and conduct process performance and efficiency analysis. We identify bottlenecks, define and detect basic and composite anti-patterns. In addition to control flow analysis, we mine event log to perform organizational analysis and discover metrics such as handover of work, subcontracting, joint cases and joint activities.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*process metrics, performance measures*; D.2.9 [**Software Engineering**]: Management—*life cycle, programming teams, software process models*

General Terms

Algorithms, Measurement, Management, Performance

MSR'14, May 31 – June 1, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2863-0/14/05...$15.00
http://dx.doi.org/10.1145/2597073.2597081

Keywords

Process Mining, Software Maintenance, Issue Tracking System, Peer Code Review System, Social Network Analysis, Empirical Software Engineering and Measurements

1. RESEARCH MOTIVATION AND AIM

Issue tracking systems such as Bugzilla[1] and Mantis[2] are applications to track and manage issue reporting and resolution. Peer code review systems such as Gerrit[3] and Rietveld[4] are web-based tools to facilitate code inspection and peer code review to avoid defects before they are injected into the system. ITS, PCR and Version control systems such as SVN[5] and Mercurial[6], are workflow management systems jointly supporting the bug reporting and resolution process in software maintenance. Free-Libre/Open Source Software (FLOSS) projects like Google Android and Chromium follow a process consisting of bug reporting in issue tracking system, patch submission for review in peer code review system in order to resolve the reported issue and committing source code change using version control system. Bug reporting and resolution process spanning across three workflow management systems generates process and event log data which is archived in these software repositories.

Process mining is an area at the intersection of business process intelligence and data mining consisting of mining event logs from process aware information systems for the purpose of process discovery, process performance analysis, conformance verification, process improvement and organizational analysis. Process mining software repositories 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 are: uncovering runtime process model [4][7], discovering process inefficiencies and inconsistencies [1][4], observing project key indicators and computing correlation between product and process metrics [13], extracting general visual process patterns for effort estimation and analyzing problem resolution activities [8]. The broad research motivation of the work presented in this paper is to investigate the application of simultaneous process mining three software repositories

[1]http://www.bugzilla.org/
[2]http://www.mantisbt.org/
[3]http://code.google.com/p/gerrit/
[4]http://code.google.com/p/rietveld/
[5]http://subversion.apache.org/
[6]http://mercurial.selenic.com/

(ITS, PCR and VCS) for deriving insights and patterns useful for project teams. We conduct a case study on Google Chromium project data to demonstrate the effectiveness of the proposed approach and applications. Specific research aims of the work presented in this paper are the following:

1. The information management systems like ITS, PCR and VCS are not explicitly linked. One of our aim is to investigate regular expression based approaches to extract textual information from bug report description and comments, and peer code review description and commit log message, for the purpose of integrating three standalone repositories. Thus we can perform simultaneous process mining across ITS, PCR and VCS.
2. Popular FLOSS projects like Google Chromium and Android do not define any design time process model for the end-to-end bug resolution process involving ITS, PCR and VCS. Our research aim is to define activities (events) pertaining to three systems, create an event log and apply process discovery algorithms to uncover runtime process model for investigation from control flow and organizational perspective.
3. To process mine event log generated from three repositories from control flow perspective (also known as process perspective [16]): performance issues, event distribution, most frequent and unique traces, bottlenecks and discovering anti-patterns.
4. To investigate application of mining event log from organizational perspective by applying social network analysis techniques to discover various interesting patterns and metrics like: handover of work, subcontracting, working together (joint cases) and joint activities.

2. RELATED WORK AND RESEARCH CONTRIBUTIONS

In this section, we discuss closely related work and present the novel research contributions of this research paper in context to the existing work. We organize closely related work into following three lines of research:

2.1 Process Mining Software Repositories

Gupta *et al.* present a framework for mining bug report history for discovering process inefficiencies and inconsistencies. They mine Bugzilla issue tracking system of Mozilla Firefox and Core project to study self-loops, back-and-forth, issue reopen, bottlenecks and present an algorithm and metrics to compute the degree of conformance between the design time and the runtime process [4]. Sunindyo *et al.* propose an observation framework that supports OSS project managers in observing project key indicators such as checking conformance between the designed and actual process models [13]. Kindler *et al.* present an approach of mining user interaction data with a document version control system for automatic derivation of a descriptive process model [7]. Knab *et al.* present an approach of extracting general visual process patterns for effort estimation and analyzing problem resolution activities for ITS [8].

2.2 Simultaneous Process Mining of Multiple Workflow Management Systems

Poncin *et al.* present a framework called as FRASR (FRamework for Analyzing Software Repositories) that facilitates combining and matching of events across multiple repositories [10]. They present an approach to combine related events (instances in multiple repositories belonging to the same case or process instance) spanning across multiple software repositories like mail archives, subversion and bug repositories followed by assignment of role to each developer [10]. Song *et al.* apply process mining technology to common event logs of information systems for behavior pattern mining [12]. They create one input data for behavior pattern mining from event logs of five different information systems [12]. Kim *et al.* propose a distributed workflow mining approach to discover workflow process model incrementally amalgamating a series of vertically or horizontally fragmented temporal work-cases [6].

2.3 Application of Social Network Analysis for Mining Software Repositories

Dittrich *et al.* model a software project as a network using information mined from the project's version control repository and apply network analysis techniques to identify the key authors and subject matter experts [2]. Sureka *et al.* mine defect tracking system to derive a collaboration network and apply social network analysis techniques to investigate the derived network for the purpose of risk and vulnerability analysis [14]. Meneeley *et al.* improved the developer activity metrics to measure the developer's collaboration by introducing two issue tracking annotations [9]. Sarma *et al.* developed Tesseract, an interactive exploratory environment to enable study of complex socio-technical relationships between different project entities [11]. Wolf *et al.* examined task-based communication and collaboration in software teams and studied the communication pattern to identify causes of build failure [17].

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

1. While there has been work done in the area of process mining single and multiple software repositories, the study presented in this paper is the first work on *simultaneous process mining of three (ITS, PCR and VCS) software repositories from both control flow and organizational perspective.*
2. While there has been several studies on applications of social networking analysis on data, this paper presents the first study on *process mining from organizational perspective to extract team-based interaction patterns, visualization and metrics* such as handover of work, subcontracting, working together and joint activities.
3. We conduct an *in-depth empirical analysis on Google Chromium project (open-source) data extracted from Google issue tracker, Rietveld peer code review system and SVN version control system* to demonstrate the application and effectiveness of our approach. We present results on runtime process model discovery, efficiency analysis and anti-patterns as well as from organizational perspective.

3. RESEARCH METHODOLOGY AND FRAMEWORK

We conduct experiments on dataset downloaded from issue tracking system (Google ITS), peer code review system (Rietveld) and version control system (Subversion) of Google Chromium browser project. It is a large, long-lived and complex open source software project. Issue reports, patches and

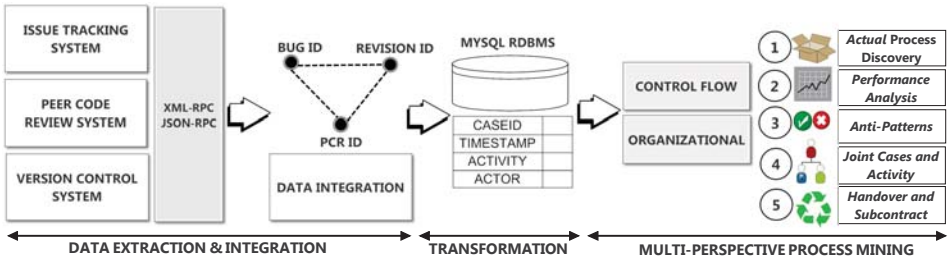


Figure 1: Research framework for process mining three software repositories from Control Flow and Organizational perspective.

commit details for Google Chromium browser are publicly available hence, the experimental analysis presented in this paper can be replicated by other researchers and used for benchmarking and comparison. Figure 1 depicts the multi-step framework adopted for this research work. It has mainly three steps: data extraction and integration, transformation and multi-perspective process mining. During the first step, we extract data from multiple software repositories using JSON-RPC and XML-RPC. Since the data is extracted from multiple information systems which are not explicitly linked, integration of extracted data is performed which is a technical challenge. The extracted data is integrated using textual analysis and regular expression matching, for mapping three distinct systems. As shown in Figure 1, the integrated data is stored in MySQL relational database which is transformed to the format suitable for process mining. An event log is generated with four fields: caseID, timestamp (ts), activity and actor. Transformed data can be process mined from multiple perspectives [16] like process (or control flow), organizational and case perspective etc. However our focus is on control flow and organizational perspective which is presented in detail and includes runtime (reality) process discovery, performance analysis, anti-patterns identification, analysis for joint cases, joint activities, handover of work and subcontracting.

Table 1: Experimental dataset details (Chromium Project).

Attribute	Value
First ITS issue creation date	1 Jul 2011
Last ITS issue creation date	30 Jun 2012
Total extracted closed ITS issues	35035
ITS issues with patches in PCR	10110
ITS issues with PCR issue reports authorized for access	10000
Total PCR issues for above ITS issues	19952
Unique PCR issues for above ITS issues	17979
Total VCS commit (out of all PCR issues)	19422

3.1 Data Extraction and Integration

We extract one year data of Google ITS starting from 1 July 2011 to 30 June 2012 using Google issue tracker APIs (refer Table 1). Data for 35035 issue reports is extracted and we find that all the extracted issues are closed. Some issues reported in Google ITS require source code change (patch) for resolution which are peer reviewed to avoid defects before they are committed into the source code. Our focus is to study the process followed from the inception (issue reported in ITS) till resolution for the issues requiring code changes. We identify such issues by performing textual analysis of comments. We start with ITS issue report and map to PCR by detecting the presence of code review system URL[7] [8] in comments marked as step 1 in Figure 2a. We obtain PCR issue ID from the links posted in comments which facilitates mapping to PCR issue depicted as label 2 in Figure 2a followed by extraction of PCR patch report details. There are some ITS issues with code review URLs in description which we do not consider for our analysis as we observe that review URLs in description are mostly included to point the change which caused the bug. Around 29% (10110) of the total ITS issues have link to PCR system in comments. We extract PCR issue ID from all the comments of these ITS issues and extract PCR report details for the same. Description of PCR issue has *BugID* to point the ITS issue for which the patch is submitted however, there are cases when *BugID* is not mentioned in description even though the patch fixes a bug. Therefore based on manual inspection, we consider all PCR issue IDs derived from comments as valid that is, intended to resolve the bug. We find that PCR issue details are accessible for 10000 ITS issues as mentioned in Table 1. One ITS issue can have one or more PCR patch reports associated with it.

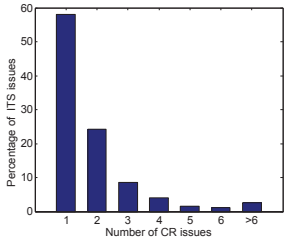
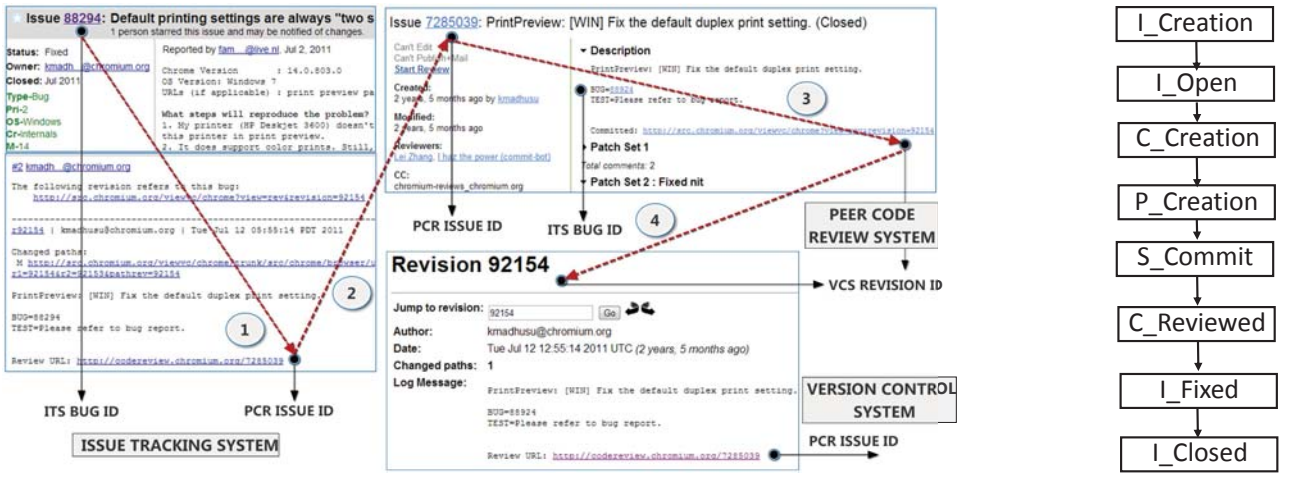


Figure 3: Distribution of number of CR issues for ITS issues.

As shown in Figure 3, most of the ITS issues (around 58%) have only one PCR issue associated with them and few have more than 6 also. Overall, we extract 17979 unique PCR patch report details and observe that some patches address more than one issue. Thus the total mapping count from ITS to PCR is 19952 which clearly shows that the relation between ITS and PCR is *Many-to-Many*.
Patch report has a brief description and an initial patch submitted by author while raising an issue in code review

[7] codereview.chromium.org
[8] chromiumcodereview.appspot.com



(a) Snapshot for mapping three information systems: ITS, PCR and VCS.

(b) Process map

Figure 2: Illustration to map multiple software repositories and corresponding process map.

system. The issue is assigned to reviewer(s) for review to detect maximum possible defects for better code quality and maintainability. Subsequent patches are submitted for same issue to address the comments of reviewers. Code change is committed to source code after the approval of reviewers and corresponding unique revision ID is generated in VCS (subversion). However if patch is not approved by the reviewers then PCR issue is closed without commit. Therefore mapping between PCR and VCS is *One-to-Zero* if closed without commit otherwise *One-to-One*, if patch set committed successfully as unique VCS commit. As depicted in step 3 of Figure 2a, PCR issue description has "Committed:" as part of its text where the link to VCS is posted after commit. There are very few exceptions where the commit link is not posted which we ignore. We extract revision ID details from PCR issue report description and use it for mapping to VCS (labeled 4 in Figure 2a). We find 19422 out of 19952 PCR issue reports have link to VCS posted in description. Commit data for all the derived revision IDs is extracted from VCS.

After extraction and integration of data from all the three repositories, we transform it to make it suitable for multi-perspective process mining.

3.2 Data Transformation

One of the major challenges in software process mining is to produce a log conforming to the input format of process mining tool [10]. Therefore, data transformation is very crucial before analysis with a process mining tool. Following are the activities performed during transformation:

- *Event log generation:* Table 2 presents a list of identified important activities, role of person performing the activity and significance of activity for each information system (IS). There are 8 distinct activities for ITS, 3 for PCR and 1 for VCS. We believe that listed activities are the important ones capturing the progression of a bug during its life cycle and have impact on overall process performance. Code review system activities are distinguished by a sequence number showing the order of PCR issue occurrence in ITS like if it is third PCR issue raised to resolve same ITS issue, sequence number 3 is added as suffix thus activity is *C_Creation3*. Similarly, activities (like *P_Creation, V_Commit*) pertaining to given PCR issue are suffixed with the same sequence number. Event log conforming to input format of process mining analysis framework (like Disco) is generated with 4 attributes: *caseID*, *activity*, *timestamp* and *performer* (actor) for all the 3 systems. These fields are derived from extracted data where *caseID* is basically the ITS issue ID, *activity* is one of those listed in Table 2, *timestamp* is the time when an activity is performed and *performer* is mail ID of a person performing the activity. For example, the mail ID of issue reporter is stored in performer field of *I_Creation*. Every activity has a single performer except *C_Reviewed* for which performer is multi-valued as one PCR issue can be reviewed by multiple reviewers. Since the activities involved in life cycle of an issue span across three systems, we interlace transformed event log from three systems such that the activities pertaining to same ITS issue are identified by same *caseID* (ITS issue ID).
- *State minimization and cleaning:* We replace status *Started, Untriaged, Available, Assigned, Unconfirmed, Accepted* (part of ITS labels for Open status of a bug[9]) with a common activity *Open* as this minimization will simplify the control flow analysis without missing any useful information. Here different labels for open status do not capture substantial progress towards resolution so we do not depict them separately however, if required one can use them as it is. We find very infrequent occurrence of status labels like *IceBox, Blocked, Moved, Upstream, FixUnreleased, WillMerge, ExternalDependency* for ITS issues (less than 10 cases). We remove these events from the final event log to ensure that the derived process model is not unnecessarily complex.

An instance is presented in Figure 2b where the activities of derived event log are ordered according to increasing timestamp. It depicts control flow for an instance used to explain the integration of three repositories. Here an issue

[9]http://www.chromium.org/for-testers/bug-reporting-guidelines

Table 2: List of activities, its significance, role of performer for all information systems.

Information System	Activity	Description	Role
Issue Tracking System	I_Creation	Issue reported in ITS	Bug Reporter
	I_Open	Open bug status label	Bug Owner, Triager, QA Manager
	I_Fixed	Resolved as Fixed	Bug Owner
	I_Invalid	Illegible, spam, etc.	Bug Owner, Triager, QA Manager
	I_Duplicate	Similar to other issue	Bug Owner, Triager, QA Manager
	I_WontFix	Can't repro, Working as intended, Obsolete	Bug Owner, Triager, QA Manager
	I_Verified	Resolution verified	QA Manager
	I_Closed	ITS progress ends	Bug Owner
Code Review System	C_Creation	Initial patch reported in PCR	Patch Author
(suffix: seq. no.)	P_Creation	Subsequent patch submit with corrections	Patch Author
	C_Reviewed	Code review process ends	Patch Reviewer
Version Control System	V_Commit	Code change committed	Patch Committer

is reported followed by patch creation, commit, completion of review and finally closed after getting fixed. Similarly, the event log obtained after transformation for one year issues can be used to obtain a general runtime process model and for process mining from multiple perspectives. Final transformed event log used for experiments is made public[10] (details in uploaded README file).

4. CONTROL FLOW PERSPECTIVE

Control flow (or Process) perspective helps to identify the activities performed and the order of their execution. There is no process defined spanning across three repositories for bug resolution. The automatically obtained process model from event log is an alternative to manual process design. Process model obtained from event log reflects the reality and can be refined for further improvements instead of defining a new design model from scratch. We have process defined for code review [5] and bug life cycle[9], which are actually subprocesses if we look at complete end-to-end life cycle of a bug from inception to resolution and it is important to understand control transfer between the subprocesses. There are many process mining tools such as ProM (open source) and Disco (commercial for which we obtained academic license) used to derive process model. We import preprocessed data into Disco to discover runtime process model from actual event log generated during the progression of a bug and other statistical information for process spanning three software repositories. Disco miner is based on the proven framework of Fuzzy Miner with completely new set of process metrics and modeling strategies[11]. ITS issue ID is selected as case (for process map generation in Disco) to associate all activities pertaining to same issue ID so that we can visualize the complete life cycle of a bug and control transfer between different systems. We perform statistical analysis of derived process map for event frequency, unique traces and bottleneck identification (time perspective). Also we define anti-patterns for our process and detect their existence in discovered runtime process map.

4.1 Process Map Discovery and Analysis

We import an event log for 9744 cases (with < 7 code review issues) to Disco for process map generation. There are 32 nodes in generated process map as shown in Figure 4 (after References due to spacing) each corresponding to an activity. Out of 32, there are 8 activities from ITS and remaining 24 ($6 * 4$) generated from 4 unique PCR and VCS activities (with 6 distinct sequence numbers). Process map presented in Figure 4 shows core transitions (for simplicity and clarity) however, the process model at 100% resolution (with edges for infrequent transitions) is very complex and looks like spaghetti. Label on edges represents absolute frequency of transition and value in an activity node is the total number of times that activity is performed in complete event log. Shade of a node and thickness of an edge in process map corresponds to absolute frequency of activity and transition respectively. We make following observations from discovered process map:

1. It is evident from discovered process map (refer Figure 4) that *I_Creation* is first activity for majority of the cases (8349 out of 9744) which means that an issue is reported in ITS followed by patch submission to PCR in order to resolve an issue. However 1356 cases have *C_Creation1* as first activity indicating that a patch is first submitted to PCR followed by an issue creation in ITS. We observe that comparatively high that is, 55% (unlike for all the cases with 40% of them having more than one code review issue as shown in Table 3) of cases with delayed ITS issue report have more than one PCR issue associated with them. We believe that ITS issue is reported for such cases after the patch author realizes that only one code review issue may not be sufficient for the fix hence issue reporting becomes crucial for traceability. It is recommended and is a good practice to first report an issue in ITS before submitting a patch to PCR in order to maintain a complete record of code changes with reason[12]. However we have identified a good number of cases (1356) where this practice is not followed. Remaining 39 cases have unexpected behavior in which the patch is directly committed to VCS without review followed by issue creation in PCR and ITS. These are very infrequent cases caused due to some critical project fix by committers.
2. The number of cases reduce with an increase in PCR sequence number as shown in Table 3. Majority of the issues are resolved with one patch issue report to PCR. Out of 9744 cases, less than 1000 cases (ITS issues) have more than 3 code review issues associated

[10]https://github.com/Mining-multiple-repos-data/experimental_dataset

[11]http://fluxicon.com/disco/files/Disco-Tour.pdf

[12]http://www.chromium.org/developers/contributing-code

Table 3: Distribution for the number of ITS issues with given CR issue sequence number and total committed cases.

CR sequence no.	Total cases (% of total issues)	Committed cases (% of total cases)
1	9744 (100)	9428 (96.7)
2	3945 (40.5)	3851 (97.6)
3	1528 (15.7)	1494 (97.7)
4	678 (6.9)	663 (97.7)
5	284 (2.9)	277 (97.5)
6	123 (1.3)	123 (100)
Total	16302	15836 (97%)

with them. Only 123 cases have 6 code review issues. Around 97% of total PCR issues (16302) are committed to VCS. Surprisingly, a good number of total committed patches (around 42% of total PCR issues with transition to *V_Commit* directly from *C_Creation*) do not need corrections and are approved for commit without any changes. This indicates that the quality of reported patches is quite good.

3. As observed from Figure 4, code review issues are created sequentially one after the other. Patch author reports a patch which is committed to VCS. Sometimes an issue is not resolved with the committed patch so another patch is reported for same issue. Since the need for more patches can be realized only after review and commit of current patch, patches for same ITS issue are reported sequentially.
4. Interestingly, frequency of *I_Fixed* is very high (89% of cases resolved as *Fixed*) which is desired and contributes towards overall software quality improvement. If source code is changed to resolve an issue, it is highly likely to get fixed as code change is an indicator of more careful involvement to address reported issue. Also, evident from our analysis as 89% of ITS issues with code change are resolved as *Fixed*. There are cases with final resolution not as *I_Fixed*, so efforts are wasted in changing the source code and review for less important issues. Therefore if a bug is duplicate or invalid, it is good if identified early to atleast avoid unnecessary code change and review efforts. We notice that the number of such cases is very less hence, the quality of issue resolution is fairly good.
5. Some issues are initially marked with wrong status as *Duplicate* or *WontFix* or *Invalid*, which are *Fixed* later. It happens if additional information becomes available to fully understand an issue or the initial status was assigned incorrectly. It emphasizes the need to fully understand an issue to reduce wrong label assignment and extra delay in reopening wrongly closed issues.
6. Final bug resolution is verified for only 35.5% of cases which is significantly less signaling inefficiency. It highlights the need to identify reasons for infrequent verification and address them. Efforts should be made to ensure that more closed bugs are verified.

Event Frequency: There are 122007 events in complete log. Number of events per case ranges from 5 to 91. Majority of the cases have 6 to 13 events as depicted in Figure 5. Number of events goes high in some cases because of subsequent patch submissions (*P_Creation* occur multiple times for PCR issue) to address reviewer comments and make patch suitable for commit. Careful review process prevents defects from getting into the software. Process is efficient from the perspective that majority cases are resolved in around 13 major events with some cases having more events showing the rigor and involvement of contributors.

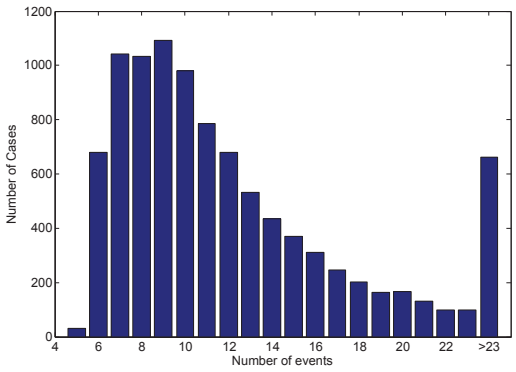


Figure 5: Number of cases with total events in life cycle.

Unique Traces: Complete sequence of events starting from first event to last event for a case is known as trace. It captures the order of activities performed for an issue. We consider two traces different even if one transition varies. For example, if an issue has 2 patches submitted during review and other has 3 then they are considered as different. We have 4453 unique traces for our dataset, which is fairly large. However the distribution is skewed with some traces being more frequent as 50% of the cases are covered with only top 6% of unique traces. Most frequent trace covering 4.39% of cases is: *I_Creation* → *C_Creation1* → *V_Commit1* → *C_Reviewed1* → *I_Fixed* → *I_Closed*, which is minimum set of activities to successfully fix an issue.

4.2 Bottleneck Analysis

A bottleneck is defined as a specific part within the runtime process map which is relatively time consuming and delays overall process [4]. This analysis from time perspective is important for process analyst to identify the cause of delay from as-is process and hence take corrective actions to improve process efficiency. We employ Disco process mining tool to calculate mean-time between the transitions and detect potential causes of delay.

We notice that the transitions between code review system and ITS are most time consuming. Discovered process map and mean time between transitions reveals that there is a significant delay in passing control from ITS to PCR and vice-versa. For instance, time for reporting first code review issue after last ITS activity is around 8 days. Similarly it takes 7 days to create second code review issue after closing of the first one. Also whenever control is transferred back to ITS from PCR, it takes comparatively more time. As observed from discovered process map, it may take as high as 8 days to mark an issue *I_Fixed* after the completion of patch review (if marked after *C_Reviewed2* and *C_Reviewed3*). Whereas two consecutive activities within PCR are happening in usually less than 24 hours. Therefore efforts should be made to ensure smooth and quick transition between ITS and PCR.

Also delay in verifying an issue after final resolution as *Fixed* is very high, that is, around 50.4 days. So QA managers should make more efforts to verify maximum ITS issues soon after they are closed.

4.3 Anti-patterns

Anti-patterns represent erroneous interdependencies between process models. Eid-Sabbagh *et al.* define basic, composite and nesting anti-patterns [3]. We identify anti-patterns (undesired transitions) and cause of their existence for our process model. It is important to analyze anti-patterns for a process analyst to detect and eliminate erroneous transitions thus, improving the overall process quality. We denote ITS process as P, PCR process as Q and VCS commit as R.

4.3.1 Basic Anti-patterns

We study anti-patterns exposing incorrect behavior confined to single information system. Gupta *et al.* [4] discuss presence of loops, back-forth and reopening patterns for ITS which are undesired (anti-patterns). We notice presence of loops in *P_Creation* activity of sub-process Q as prominent anti-pattern. Figure 4 reveals 6160 cases have patch submission (*P_Creation1*) after initial patch with some cases having as high as 59 subsequent patches. Similarly for *P_Creation2* out of 3945 cases, 1442 cases have subsequent patches with some cases having 37 patches where each subsequent patch submission has mean delay of 20.7 hours. Subsequent patches are submitted as a result of careful review by reviewers however, it delays the process as every subsequent patch has mean delay of 27.9 hours (in case of *P_Creation1*). It is undesired to have very high number of subsequent patch submissions. Therefore, efforts should be made to minimize delay without compromising the quality of review.

4.3.2 Composite Anti-patterns

We study anti-patterns involving triggering and information flow between two or more processes. P is parent process and Q is sub-process for issue resolution, that is, Q should be triggered by the need of code change to resolve an issue reported in ITS. Thereafter once the patch is accepted, it should be committed to VCS (R) followed by *C_Reviewed* marking the completion of sub-process Q and subsub-process R. The control is transferred back to P for further progression towards issue resolution. Therefore at process level $(P,Q) \in triggers^{as}_{int}$, which means P triggers Q asynchronously (as) with an intermediate (int) event of patch creation and $(Q,R) \in triggers^{as}_{int}$ with an intermediate step of patch acceptance [3], both should hold true. It is asynchronous because PCR issue need not be instantiated the moment when need for code change is realized, it can be done after sometime. Similarly a patch can be committed to repository anytime after it is accepted.

However there are cases when P is instantiated after Q and also R is not triggered after Q that is, above mentioned conditions are violated. For fairly large number of cases 1395 (14.3%), Q or R are behaving as parent process. It implies that the requirement to maintain an issue in ITS is triggered from the intermediate activity of process Q or R. One reason could be: initially the patch author aims to fix an issue by directly submitting a patch to PCR however, author realizes the complexity of issue and need for more than one patch thus, an issue is reported in ITS for traceability. Sometimes it may be because the fix is very critical so a patch is first committed followed by issue creation in ITS and PCR just for the record purpose. This situation of emergency happens very rarely therefore, efforts should be made to ensure that an issue is first reported in ITS followed by further progress spanning across PCR and VCS in order to resolve an issue.

5. ORGANIZATIONAL PERSPECTIVE

Process mining from organizational perspective aims to identify organizational roles, work distribution among performers (resource) and activities that can be executed by a particular resource. Open source projects like Google Chromium are driven by volunteer contribution and is important to investigate the interaction pattern between various contributors to improve the efficiency of project. We build social networks based on relations like causality (handover and subcontracting of work), joint activities and joint cases using the event log data [15]. We use *Gephi*[13], an interactive visualization platform to visualize the social network graphs and perform social network analysis.

5.1 Joint Cases

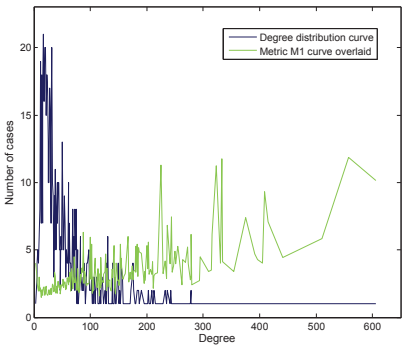


Figure 6: Degree distribution graph with metric M1 curve overlaid for analysis.

We study interaction pattern between various organization members by plotting degree distribution graph and evaluate defined metric $M1$ for strength of interaction. Individuals working together (on same case) will have a stronger relation than the people rarely working together [15]. We consider only the actors with participation in more than 3 cases. Degree distribution curve is plotted in Figure 6 where horizontal axis represents degree and vertical axis is the total number of actors having that degree. We observe from degree distribution curve of Figure 6 that most of the actors have degree upto 50 with some having more than 100 also. Number of actors decreases with an increase in degree, also verified statistically. The Pearson correlation coefficient is 0.5575 with negative sign indicating relation between the reduction in number of actors with an increase in degree. It means that most of the actors work with comparatively less people and few have a large social circle. We evaluate and plot metric $M1$ for each degree d as:

For each vertex with degree d,

$$M1 = \frac{\sum_{i=1}^{N}(w_i)}{d * N}$$

[13]https://gephi.org/

where N = Number of vertices with degree d, and
w_i = Weighted degree of vertex i having degree d
Metric $M1$ measures average strength of interaction for vertices (actors) having degree d with its neighbors. Green line in Figure 6 represents value of metric $M1$ for each degree. We notice from Figure 6 that as the degree increases, strength of interaction also increases which implies that in general the actors working with diverse people also work with them for more number of times. The correlation coefficient between degree and weighted average degree ($M1$) is 0.6661 (fairly high) showing a positive relation between them. It supports the observation that the strength of interaction is also high for individuals with large social circle. Effectively they are *active contributors* as working on multiple cases together with multiple people (both weighted degree and degree are high). We calculate relative working together metric [15] for performer *p1* and *p2* using complete log L as:

$$p1 \bowtie_L p2 = \frac{Number\ of\ cases\ p1\ and\ p2\ worked\ together}{Number\ of\ cases\ p1\ participated}$$

Table 4: List of top 5 instances working together with relative working together metric values.

p1	p2	Cases together	$p1 \bowtie_L p2$	$p2 \bowtie_L p1$
A	B	358	0.094	0.053
C	B	337	0.093	0.050
D	B	314	0.080	0.047
C	A	256	0.070	0.067
D	C	249	0.063	0.068

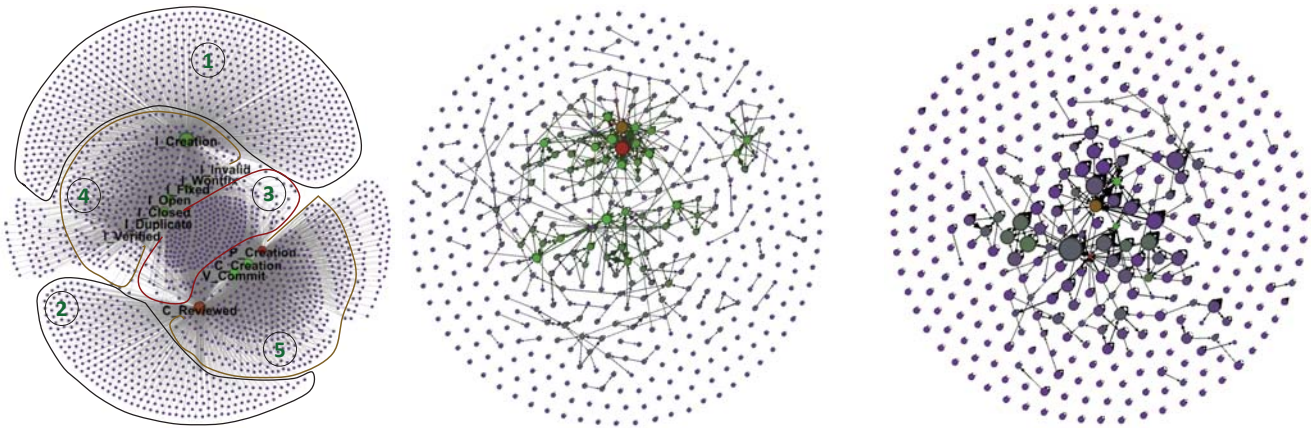
A list of top five most frequently working together actor pairs (instances) with value for relative working together metric is presented in Table 4. Performer names are aliased for anonymity. Interestingly we notice that active members interact more with active members. We observe that A is twice closely related to B than B's relation to A because out of all the cases on which A works, he relatively works more often with B. Similarly for C-B and D-B the relative working together metric shows high tendency of C and D to work with B. However for C-A and D-C, both the actors equally tend to work with each other. Therefore we can recommend a group of people to work on same case using relative working together metric as it gives information about the strength of relation between different people.

5.2 Joint Activities

We create an adjacency matrix where row represents list of actors and column as activities they perform. Value in each cell corresponds to the frequency of an activity performed by a particular performer. Figure 7a depicts relation between performers and the activities they perform where size of each vertex is proportional to degree and color ranges from blue (minimum), green (medium) to red (maximum) corresponding to weighted degree. If an actor performs an activity then an edge is drawn between actor and the activity. There are total 2160 unique performers involved in 9744 cases. We notice from Figure 7a that total 1236 unique reporters report an issue in ITS. A major section of performers (marked with label 1) reporting issues in ITS is isolated that is, large number of actors only report issues and do not participate in any other resolution activity. Here we have diverse performers reporting issues to ITS for less number of times. Total 915 unique performers review patches and we notice that majority of reviewers do not participate in any other activity. Reviewers should preferably be different from developers which is the case thus, indicating efficient task distribution. Oval 2 in Figure 7a highlights a big group of reviewers who are specialists in reviewing the patches. We can identify a group of actors reviewing patches more frequently using available social graph with high weighted edges and give them commit rights to improve overall process performance and role assignment. There are comparatively less contributors performing more than one role that is, *generalists*. Hence group of *generalists* and *specialists* is identified for efficient task allocation. We observe that code review activities have comparatively high weighted degree (nodes with bright green and red shades). This implies that more distinct people engaged with ITS whereas less, dedicated people involved with patch submission and review activity.
We observe that group of actors who participate in all the three systems is very small, labeled as 3 in Figure 7a. Majority of the actors contribute to only one system by performing single or multiple activities confined to the same system (refer label 4 and 5 in Figure 7a). Therefore a small group of contributors (labeled as 3) is very crucial as they have knowledge of multiple systems and are *core generalist* contributors. It highlights the need to encourage more people for participation in multiple activities and make them aware of different information systems involved in overall issue resolution process.

5.3 Handover of Work

Handover of work is based on the idea that two performers are related if a case is passed from one performer to another [15]. We consider only direct succession as handover that is, an activity executed by performer $p1$ is consecutively followed by an activity executed by performer $p2$. Also multiple transfers between same performers (same instance) are counted and added to frequency. We filter instances with handover frequency less than 15 to remove infrequent instances (not important for our analysis of frequent handover identification). We identify 988 unique handovers including self-loops having frequency more than the threshold (15). There are 472 vertices in the sociogram obtained for handover as shown in Figure 7b. Out of 988, most of the instances (429) are self loops which means that a good number of performers perform subsequent activities more than 15 times. We notice many isolated vertices as they only have self loop and no frequent handover with other performers. Size of vertex corresponds to out-degree and color varies with in-degree where blue is for low, green is for medium and red corresponds to high range of in-degree. We observe from Figure 7b that nodes with high in-degree have high out-degree as well with few exceptions where in-degree is comparatively greater than out-degree that is, small sized node with green color. Therefore there is no performer with highest authority (out-degree significantly greater than in-degree) to only handover work to others which is in adherence with expectation for an open source project where the contributors are volunteers and take up tasks of their choice. Highest weight edges are self loops, where weight is the frequency of handover, supporting continuous task execution by same performer. There are instances with handover between different individuals with frequency as high as 221 and 197. Also we observe in some instances that the edge is bi-directional implying



(a) Social network representing relation between activity and performers (resources).

(b) Sociogram representing handover of work between performers where size of node is proportional to out-degree, color shade corresponds to in-degree ranging from blue (minimum) to red (maximum).

(c) Sociogram representing subcontract of work between performers where size of node is proportional to out-degree, color shade corresponds to in-degree ranging from blue (minimum) to red (maximum).

Figure 7: Analysis from Organizational Perspective

the handover of work both ways. However, there are many instances where the work is transferred only in one direction so we can say that only the destination vertex (second actor) follows the activity performed by source vertex (first actor). Therefore, we identify handover dependency between different performers without causality validation (if the reason for activity performance by actor $p2$ is task executed by $p1$).

5.4 Subcontracting

If performer $p1$ performs an activity followed by $p2$ then again $p1$, it is considered as subcontracting of work to $p2$ by $p1$ (directed edge from $p1$ to $p2$). We study cases with only one activity in-between two activities performed by the same performer that is, direct succession for subcontract. We consider multiple occurrences of same instance. We identify 6771 subcontract instances with most of the instances having low frequency. We filter instances with subcontracting frequency less than 11 to focus on frequent and interesting instances. Overall we have 577 instances (edges including self loop) above threshold involving 370 unique performers (vertices) as shown in Figure 7c. Size of vertex is proportional to out-degree and color of node ranges from blue (lower) to red (highest) for in-degree. There are more nodes with relatively large size in blue or green color as shown in Figure 7c indicating that these are the performers with case subcontracted to them by few other performers however, they subcontract work to more performers. There are very few nodes with red color (high in-degree) and small size(less out-degree) implying that more individuals subcontracted work to them. As observed from Figure 7c, there are many nodes (on periphery) with only self loop signifying that same performer is performing subsequent activities multiple times (atleast more than 2) with no subcontract to any other performer. Infact the frequency of self loops (weight of self loop edges) is comparatively high which means that same performer often continues working for multiple times. At the center of sociogram in Figure 7c is a chain of subcontract between different performers showing that they are more related with each other as work is subcontracted between them.

6. CONCLUSION

Process map (reality) discovered from runtime event log for 9744 Google Chromium ITS issues, with resolution activities spanning across three information systems Chromium ITS, Rietveld PCR and Subversion VCS, reveals control flow for complete lifecycle of issues. Process map with 32 states (activities) and core transitions derived which can be used as input for defining design process. Empirical analysis reveals interesting patterns, high number (4453) of unique traces and event distribution with most of the cases having $6-13$ events in the lifecycle. We infer that the resolution process is quite efficient with high chances of getting an issue fixed. We identify bottlenecks like transition between ITS and PCR, and final resolution verification, which should be addressed to further improve the overall performance. Basic and composite anti-patterns like loops, triggering and information flow are detected to guide process analyst for corrective actions. Organizational analysis for joint activities helps to identify group of generalists and specialists. We observe that more social performers are more active contributors by evaluating metric $M1$ in joint case analysis and also evaluate relative working together metric which can guide for better team creation. Handover and subcontract analysis brings out the fact that same performer performs multiple subsequent activities and few performers are related with each other because of handover or subcontract.

7. ACKNOWLEDGEMENT

The work presented in this paper is supported by Prime Minister's fellowship awarded to the first author. The author would like to acknowledge SERB, CII and Infosys Limited for their support.

8. REFERENCES

[1] Burcu Akman and O Demirors. Applicability of process discovery algorithms for software organizations. In *Software Engineering and Advanced Applications, 2009. SEAA'09. 35th Euromicro Conference on*, pages 195–202. IEEE, 2009.

[2] Andrew Dittrich, Mehmet Hadi Gunes, and Sergiu Dascalu. Network analysis of software repositories: Identifying subject

matter experts. In *Complex Networks*, pages 187–198. Springer, 2013.

[3] Rami-Habib Eid-Sabbagh, Remco Dijkman, and Mathias Weske. Business process architecture: use and correctness. In *Business Process Management*, pages 65–81. Springer, 2012.

[4] 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*. ACM, 2014.

[5] Kazuki Hamasaki, Raula Gaikovina Kula, Norihiro Yoshida, AE Cruz, Kenji Fujiwara, and Hajimu Iida. Who does what during a code review? datasets of oss peer review repositories. In *Proceedings of the Tenth International Workshop on Mining Software Repositories*, pages 49–52. IEEE Press, 2013.

[6] Kwanghoon Pio Kim. Mining workflow processes from distributed workflow enactment event logs. *Knowledge Management & E-Learning: An International Journal (KM&EL)*, 4(4):528–553, 2013.

[7] Ekkart Kindler, Vladimir Rubin, and Wilhelm Schäfer. Activity mining for discovering software process models. *Software Engineering*, 79:175–180, 2006.

[8] Patrick Knab, Martin Pinzger, and Harald C Gall. Visual patterns in issue tracking data. In *New Modeling Concepts for Today's Software Processes*, pages 222–233. Springer, 2010.

[9] Andrew Meneely, Mackenzie Corcoran, and Laurie Williams. Improving developer activity metrics with issue tracking annotations. In *Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics*, pages 75–80. ACM, 2010.

[10] Wouter Poncin, Alexander Serebrenik, and Mark van den Brand. Process mining software repositories. In *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*, pages 5–14. IEEE, 2011.

[11] Anita Sarma, Larry Maccherone, Patrick Wagstrom, and James Herbsleb. Tesseract: Interactive visual exploration of socio-technical relationships in software development. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 23–33. IEEE, 2009.

[12] Jinliang Song, Tiejian Luo, and Su Chen. Behavior pattern mining: Apply process mining technology to common event logs of information systems. In *Networking, Sensing and Control, 2008. ICNSC 2008. IEEE International Conference on*, pages 1800–1805. IEEE, 2008.

[13] Wikan Sunindyo, Thomas Moser, Dietmar Winkler, and Deepak Dhungana. Improving open source software process quality based on defect data mining. In *Software Quality. Process Automation in Software Development*, pages 84–102. Springer, 2012.

[14] Ashish Sureka, Atul Goyal, and Ayushi Rastogi. Using social network analysis for mining collaboration data in a defect tracking system for risk and vulnerability analysis. In *Proceedings of the 4th India Software Engineering Conference*, pages 195–204. ACM, 2011.

[15] Wil MP Van Der Aalst, Hajo A Reijers, and Minseok Song. Discovering social networks from event logs. *Computer Supported Cooperative Work (CSCW)*, 14(6):549–593, 2005.

[16] Wil MP van der Aalst, Hajo A Reijers, Anton JMM Weijters, Boudewijn F van Dongen, AK Alves de Medeiros, Minseok Song, and HMW Verbeek. Business process mining: An industrial application. *Information Systems*, 32(5):713–732, 2007.

[17] Timo Wolf, Adrian Schroter, Daniela Damian, Lucas D Panjer, and Thanh HD Nguyen. Mining task-based social networks to explore collaboration in software teams. *Software, IEEE*, 26(1):58–66, 2009.

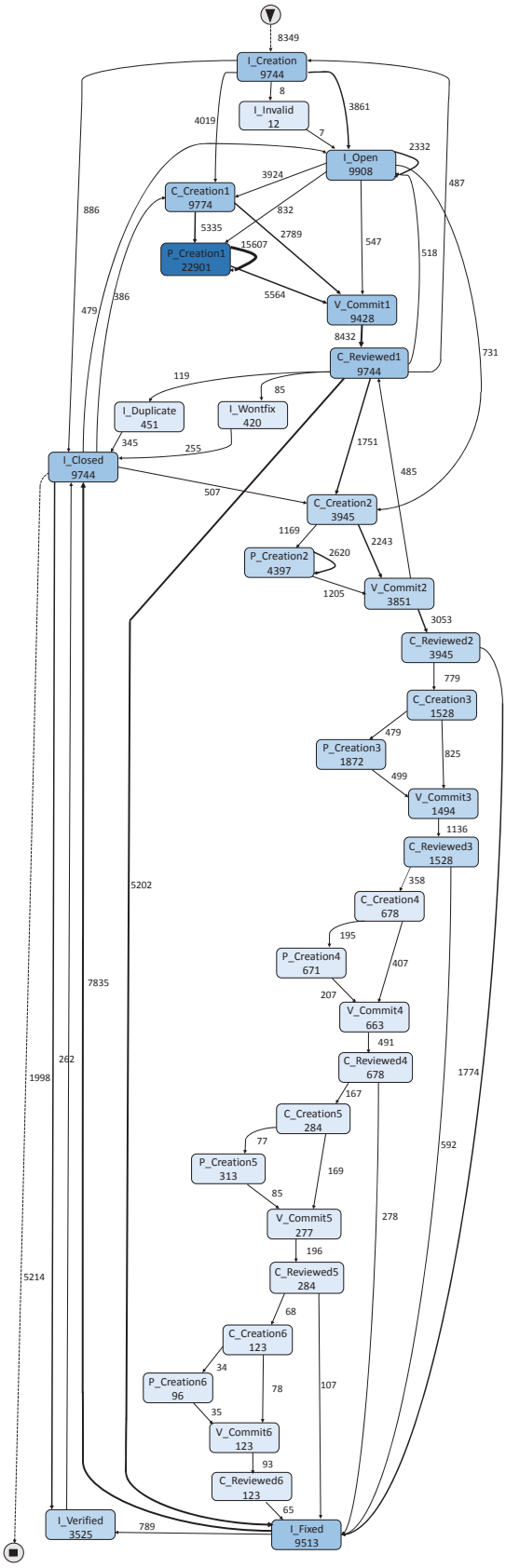


Figure 4: Process Map for Chromium bug resolution process spanning three information systems.