Process Cube for Software Defect Resolution

Monika Gupta, Ashish Sureka
Indraprastha Institute of Information Technology, Delhi (IIITD)
New Delhi, India
monikag@iiitd.ac.in, ashish@iiitd.ac.in

Abstract—Online Analytical Processing (OLAP) cube is a multi-dimensional dataset used for analyzing data in a Data Warehouse (DW) for the purpose of extracting actionable intelligence. Process mining consists of analyzing event log data produced from Process Aware Information Systems (PAIS) for the purpose of discovering and improving business processes. Process cube is a concept which falls at the intersection of OLAP cube and process mining. Process cube facilitates process mining from multiple-dimensions and enables comparison of process mining results across various dimensions. We present an application of process cube to software defect resolution process to analyze and compare process data from a multi-dimensional perspective. We present a framework, a novel perspective to mine software repositories using process cube. Each cell of process cube is defined by metrics from multiple process mining perspectives like control flow, time, conformance and organizational perspective. We conduct a case-study on Google Chromium project data in which the software defect resolution process spans three software repositories: Issue Tracking System (ITS), Peer Code Review System (PCR) and Version Control System (VCS). We define process cube with 9 dimensions as issue report timestamp, priority, state, closed status, OS, component, bug type, reporter and owner. We define hierarchies along various dimensions and cluster members to handle sparsity. We apply OLAP cube operations such as slice, dice, roll-up and drill-down, and create materialized sublog for each cell. We demonstrate the solution approach by discovering process map and compare process mining results from control flow and time perspective for Performance and Security issues.


I. RESEARCH MOTIVATION AND AIM

Stand-alone process analysis is the common way of analyzing processes in today’s process mining approaches [1]. There may be multiple variants of same process influenced by multiple factors like different working patterns, diverse business requirements and change with time (concept drift). Therefore, it is of great importance to cluster cases based on various attributes and compare process mining results [2]. This enables identification of more efficient paths and practices, thus, other process variants can be modified to improve performance. Process cube is a multidimensional data structure where events and process models are organized using different dimensions to facilitate comparison of multiple processes [1]. Process Cube View (PCV) is defined using Process Cube Structure (PCS) and Event Base (EB). PCS defines the dimensions of the cube, and EB is the complete set of attributes used for dimension mapping and materialized sublog creation. Relevant event attributes are selected as dimensions based on the type of comparisons to be performed. Since the idea of process cube is related to OLAP, we can exploit its features and perform functions like slice, dice, roll-up and drill-down.

ITCs (like Bugzilla1 and Mantis2), PCR (like Gerrit3 and Rietveld4) and VCSs (like SVN5 and Mercurial6), 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 issue reporting in ITS followed by patch submission for review in PCR (if required, to resolve reported issue) and committing source code change using VCS [3]. An issue reported in ITS is characterized by many attributes like Priority, Type of issue, Component to which issue pertains and OS. A common issue resolution process is defined irrespective of these attributes. This work is motivated by the need to compare and understand the differences in process for cases with different characteristics along various dimensions. We believe it is important to consider how differences along various dimensions produce process variants. Specific research aims of the work presented in this paper are as following:

1) To define PCV for software defect resolution process.
2) To investigate applications of OLAP operations like slice, dice, roll-up and drill-down on process cube, thus, compare process along various dimensions.
3) To discover process model using state-of-the-art algorithms for each process cell. Also evaluate and compare metrics from multiple perspectives like control flow, time (bottleneck), and organizational to identify inefficiencies and suggest modifications for improved performance.
4) To conduct a case-study on a popular FLOSS project, Google Chromium to illustrate the usefulness of proposed approach and compare process mining results across various process variants.

II. RELATED WORK AND RESEARCH CONTRIBUTIONS

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

A. Data Warehousing Environment in Software Development

Ruiz et al. present a DW environment to support the implementation of a process measurement and analysis program in a CMM Level 2 organization [4]. DW provides a centralized and

1http://www.bugzilla.org/
2http://www.mantisbt.org/
3http://code.google.com/p/gerrit/
4http://code.google.com/p/rietveld/
5http://subversion.apache.org/
6http://mercurial.selenic.com/
unified view of all projects and stores project-related data to support monitoring of software development according to the defined metrics [4]. Colaço et al. propose a data warehousing approach for software development data analysis [5]. They present a dimensional model for software change history and analyze its data using two tools: a change history miner and a corrective maintenance dashboard [5]. It is possible and viable to keep a DW focused on code construction [5]. Silveira et al. present a software process data warehousing architecture, SPDW+ as a solution to frequent, seamless, and automated capturing of software quality metrics, and their integration in a central repository for full range of analysis [6].

B. Process Mining Software Repositories

Gupta et al. present an application of integrating and process mining three software repositories (ITS, PCR and VCS) from control flow and organizational perspective [3]. They propose a framework to discover runtime process model for bug resolution process, identify bottlenecks, define and detect basic and composite anti-patterns and discover metrics such as handover of work, subcontracting, joint cases and joint activities [3]. 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 [7]. Poncin et al. present a framework called as FRASR (FRamework for Analyzing Software Repositories) that facilitates combining and matching of events across multiple repositories like mail archives, subversion and bug repositories followed by assignment of role to each developer [8]. Song et al. apply process mining technology to common event logs obtained from five information systems for behavior pattern mining [9].

C. Process Cubes

Aalast et al. formalize the notion of process cubes where events and process models are organized using different dimensions which allows interactive analysis and exploration of process [1]. Mamaliga et al. developed an initial prototype for process cubes, ProCube using the process mining framework ProM and the Palo OLAP toolset to allow comparison of multiple processes [2]. ProCube plugin in ProM creates sublogs per cell on-the-fly (unlike traditional process mining approaches) and visualizes the discovered process models, social networks and dotted charts [2].

In context to the existing work, we present the following novel and unique research contributions in this paper:

1) While there has been work done in the area of data warehousing for software development, to the best of our knowledge, the study presented in this paper is first work on application of OLAP model to define process cube notion for software defect resolution.

2) While stand-alone defect resolution process has been process mined from multiple perspectives, this paper presents a novel framework to facilitate comparison between various process variants by applying OLAP operations on process cube.

3) We conduct an in-depth empirical analysis on Google Chromium project (open-source) data extracted from Google issue tracker, Rietveld PCR and Subversion VCS to demonstrate the application and effectiveness of our approach. We compare process mining results for different issue Type such as Security and Performance. We present results for runtime process discovery, activity analysis, transition analysis and bottleneck identification.

III. Research Methodology and Framework

We conduct experiments on dataset downloaded from Google ITS, Rietveld PCR and Subversion VCS of Google Chromium browser project. It is a large, long-lived and complex open source software project. Issue reports, patches and commit details for Google Chromium browser are publicly available. Hence, the experimental analysis presented in this paper can be replicated, used for benchmarking and comparison by other researchers. Fig. 1 depicts the multi-step framework adopted for this research work. It has majorly three steps: 1. data extraction, integration and transformation, 2. process cube view and operations, and 3. process mining (process cell metrics). During the first step, we extract data for complete lifecycle of an issue reported in ITS using JSON-RPC and XML-RPC. We notice that for some cases, lifecycle spans across multiple software repositories (ITS, PCR and VCS). Therefore, the data is extracted and integrated from multiple information systems (IS). As shown in Fig. 1, the complete transformed Event Base (EB) having trace data for all cases is stored in MySQL relational database. PCS is defined and OLAP operations are performed to obtain and compare process mining results. Materialized sublog derived from EB for each cube cell can be process mined from multiple perspectives.
like control flow, conformance, time and organizational using various measures [10].

A. Data Extraction, Integration and Transformation

We extract 4 year data of Google ITS starting from 1 January 2009 to 31 December 2012 using JSON-RPC (refer Table I). Data for 177926 issue reports is extracted and we find that around 82% issues are closed. As shown in Table I, 39550 closed issues require source code change (patch) for resolution which are peer reviewed in PCR to avoid defects before they are committed into source code, VCS. Eventually, these cases have lifecycle spanning three IS which are not explicitly linked. Therefore, integration becomes very important and performed using the text analysis approach proposed by Gupta et al. [3]. We obtain a complete EB, that is, trace for all the cases with two category of attributes:

1. **Case Attributes**: Properties associated with a case which remain constant for all the events pertaining to same case. For example: **CaseID, Type, State, Closed Status, Priority, OS, Reporter, Owner, Component and Reported Time** of an issue.

2. **Event Attributes**: Properties characterizing each event in lifecycle of an issue like **Activity, Actor (resource) and Timestamp** (when the activity is performed).

Table III displays list of all the extracted case and event properties (attributes) with description. Some case properties are not defined for all the cases, hence, NULL is assigned as value. For instance, many cases do not have a priority label associated with them, thus, we assume priority as NULL. For event attributes, we identify a set of activities to embark progress of an issue towards resolution as shown in Table II. There are 8 activities from ITS. We group all the open status issues into three IS, which we denote as I.Open state because these labels do not mark significant difference in progress. If a patch is submitted to PCR system, we denote it as I.Create and once the patch is reviewed, we represent it as I.Reviewed.

B. Process Cube View

Process cube links the process mining framework to the existing OLAP technology. It is a multidimensional structure built from event log data for efficient process mining analysis and comparison [1]. Unlike OLAP where each cell is a numerical measure, process cube cell consists of a set of events derived from EB based on dimension values. We identify dimensions for defect resolution PCS. All the properties, D stored in EB can be selected as dimension, Dsel for process cube. However, this may lead to inefficient performance due to sparsity. Sparsity is a challenge because of dimensions with very large domain leading to highly infrequent occurrence of its values in complete EB. This results in missing event sublog for majority of the process cells after applying various OLAP filtration operations, consequently, degrades the performance. We can handle sparsity by following ways:

- **Dimension Subset Selection**: Dimensions of final process cube can be a subset of all the characterization properties stored in EB, that is, Dsel ⊆ D. However, it is not always a viable solution to remove attribute causing sparsity from cube dimension because it involves many storage and performance challenges [2].

- **Hierarchies**: With an increase in granularity of attribute value, frequency of value decreases in complete log. Accordingly, we can use values at higher level to reduce the number of distinct members per dimension. This can only be applied to dimensions with hierarchical domain. For example, from Table III, Priority is a dimension with hierarchy Class → Value where Class can be High or Low. High can have value 0 or 1 while Low can have value 2 or 3. Here we can roll up to higher level (Class) to handle sparsity.

- **Clustering**: We can group multiple values using well defined grouping criteria. Thus, reducing the number of distinct members without compromising the quality of analysis. For example, domain of Owner is very large. Therefore, we can create classes based on the duration of participation and productivity. As shown in Table III, we reduce domain of Owner dimension to

---

**Table I: Experimental Dataset Details (Chromium Project).**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>First ITS issue creation date</td>
<td>1 Jan 2009</td>
</tr>
<tr>
<td>Last ITS issue creation date</td>
<td>31 Dec 2012</td>
</tr>
<tr>
<td>Total extracted ITS issues</td>
<td>177926</td>
</tr>
<tr>
<td>Total closed ITS issues</td>
<td>145326 (81.67% of extracted)</td>
</tr>
<tr>
<td>Total closed ITS issues with lifecycle spanning 3 IS</td>
<td>39550 (27.21% of closed)</td>
</tr>
<tr>
<td>Total closed ITS issues with type Security</td>
<td>2271 (1.56% of closed)</td>
</tr>
<tr>
<td>Total closed ITS issues with type Performance</td>
<td>716 (0.49% of closed)</td>
</tr>
</tbody>
</table>

**Table II: List of Activities, its Significance.**

<table>
<thead>
<tr>
<th>Activity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L_Creation</td>
<td>Issue reported in ITS</td>
</tr>
<tr>
<td>L_Open</td>
<td>Open bug status label</td>
</tr>
<tr>
<td>Fixed</td>
<td>Resolved as Fixed</td>
</tr>
<tr>
<td>Invalid</td>
<td>Irrebile, spam etc.</td>
</tr>
<tr>
<td>Duplicate</td>
<td>Issue has been reported in another bug, or shares the same root cause as another bug.</td>
</tr>
<tr>
<td>WontFix</td>
<td>Can’t reproduce, Working as intended, Obsolete</td>
</tr>
<tr>
<td>Verified</td>
<td>Resolution verified</td>
</tr>
<tr>
<td>L_Closed</td>
<td>ITS progress ends</td>
</tr>
</tbody>
</table>

**Table III: Activity, Actor, Resource properties.**

<table>
<thead>
<tr>
<th>Activity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_Creation</td>
<td>Patch reported in PCR</td>
</tr>
<tr>
<td>C_Reviewed</td>
<td>Code review process ends</td>
</tr>
<tr>
<td>V_Commit</td>
<td>Code change committed</td>
</tr>
</tbody>
</table>

---

<https://www.chromium.org/for-testers/bug-reporting-guidelines>

TABLE III: Case and Event Attributes in Event Base with the Description. D/ND Denotes if the Attribute is a Dimension (D) or Not Dimension (ND) of Process Cube. Domain is a Set and Type of Values Possible for Each Attribute.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>D/ND</th>
<th>Sparsity</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>CaseID</td>
<td>Unique ITS issue ID</td>
<td>ND</td>
<td>—</td>
<td>Integer</td>
</tr>
<tr>
<td>Type [11]</td>
<td>Issue type</td>
<td>D</td>
<td>No</td>
<td>Regression, Security, Crash, Performance, Usability, Polish, Cleanup</td>
</tr>
<tr>
<td>State</td>
<td>Issue is in process or closed</td>
<td>D</td>
<td>No</td>
<td>Open, Closed</td>
</tr>
<tr>
<td>Closed Status</td>
<td>Final issue resolution label</td>
<td>D</td>
<td>No</td>
<td>Fixed, Verified, Duplicate, WonFix, Invalid, External Dependency, FixUnreleased, Icebox</td>
</tr>
<tr>
<td>Priority</td>
<td>Importance of fixing an issue</td>
<td>D</td>
<td>Hierarchy</td>
<td>Class (High or Low) → Value (0 and 1; 2 and 3)</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System to which issue pertains</td>
<td>D</td>
<td>No</td>
<td>All, Chrome, Linux, Mac, Windows</td>
</tr>
<tr>
<td>Reporter</td>
<td>Person who reports issue</td>
<td>D</td>
<td>Clustering</td>
<td>Low, Medium, High</td>
</tr>
<tr>
<td>Component</td>
<td>Component to which issue pertains</td>
<td>D</td>
<td>Clustering</td>
<td>Context, Internals, Platform, UI, OS</td>
</tr>
<tr>
<td>Reported Time</td>
<td>Published time for issue</td>
<td>D</td>
<td>Hierarchy</td>
<td>Year → Month → Day → Time</td>
</tr>
<tr>
<td>Owner</td>
<td>Person who manages issue</td>
<td>D</td>
<td>Clustering</td>
<td>Long Term, Short Term Contributors: Duration and productivity based [12]</td>
</tr>
</tbody>
</table>

Event Attributes : Different for each event of a case

| Activity | Task performed | ND | — | Activities captured for issue resolution progression [3] |
| Timestamp | Time when activity is performed | ND | — | Datetime |
| Actor | Person who performs activity | ND | — | People participating in resolution process |

Fig. 2: Illustrate set of OLAP operations on sample 3-D process cube to compare process mining results for Performance and Security type issue resolution.

2 members: Long Term Contributors (LTC) and Short Term Contributors (STC) [12]. This allows for delta analysis between the issue resolution process adopted by LTC vs STC. Similarly we can classify values for other sparse dimensions like Component and Reporter according to business analysis requirements.

Based on our analysis, we conclude that it is a technical challenge to efficiently handle sparsity and select dimensions for a process cube. As observed in Table III, all the case attributes are selected as cube dimension except CaseID because it is unique so highly sparse. Also we believe that filtering based on CaseID is not very interesting for business analysis. In Table III, we present ways to handle sparse dimensions and domain for each attribute. Here we identify 7 members for dimension issue Type such as regression, security, performance, crash, cleanup, polish and usability. [11]. We notice that no event attribute is used as process cube dimension though we derive sublog with four fields: CaseID, Activity, Timestamp and Actor using event attributes from the all-encompassing EB. Step of extracting an event log from EB for process cube cell is known as materialization step. During this step, if value for case attributes belongs to the permitted set of values along filtration dimensions then we add case log to materialized sublog for a process cell. We pass derived materialized sublog as input to standard process mining tools like Disco9(commercial) and ProM10(open source) for process discovery. Also other process mining measures of interest are evaluated and compared systematically for all the cells.

C. Process Cube Operations

Process cube idea is similar to OLAP thereby, we explore application of following OLAP operations to perform diverse analysis [1]. We illustrate application of these operations on sample process cube for software defect resolution. We consider an instance cube with three dimensions: 1. Type, 2. State, and 3. Closed Status. For remaining case attributes in event base, all the domain values are allowed and no filtration constraint is applied on them. The objective is to compare process derived from execution data of different bug Type resolution.

1) Slice: Selects single value for one of the dimension, thus, reducing total number of dimensions by one. In Step 1 of Fig. 2 we slice process cube along State dimension and obtain a subcube with value for State as closed, that is, only closed issues are considered for analysis.

2) Dice: Defines a subcube by performing selection of single or multiple dimensions thereby, reducing number of members along selected dimensions. We obtain a subcube in Step 2 of Fig. 2 by selecting

9http://www.fluxicon.com/

10http://www.promtools.org/prom6/
Type as Performance and Security, and Status as Fixed, Invalid, Duplicate, WontFix and Verified. All the cases with final status (like External Dependency, FixUnreleased and Icebox grouped as Others for the sake of presentation) apart from these 5 are filtered.

3) Roll Up: Summarizes data along a dimension. For example, as depicted in Step 3 of Fig. 2 we roll up and combine all the selected Closed Status values (from Step 2). As a result, no partitioning based on final Closed Status. This reduces number of cells along the rolled up dimension.

4) Drill Down: Represents data at more specialized level of hierarchy with more number of members along the dimension. It is opposite of roll up.

Eventually, after performing sequence of above operations on sample process cube, we compare process variants for issue Type. Performance and Security. There has been research work done highlighting the importance of considering different types of bugs in software quality research and practice [11] [13]. A common issue resolution process is defined for Google Chromium project, here we aim to bring out the differences in process in-practice for different bug types. We obtain materialized sublog from complete 4 years event log data. Event log for all the cases with bug Type as Performance or Security having final Status as Fixed, Invalid, Duplicate, WontFix or Verified is added to corresponding materialized sublog for Cell 1 and Cell 2 respectively. As a result, we obtain two logs for process mining and compare from various perspectives like control flow and time using metrics proposed in following sections. We make the dataset used in our study publicly available at github11 so that our experiments can be replicated and used for benchmark purposes by other researchers.

IV. CONTROL FLOW PERSPECTIVE

Control flow perspective helps to identify the activities performed and the order of their execution. We believe it is interesting to study automatically obtained process model from the materialized sublog of Performance and Security type bugs. It reflects the reality and can be used as input and actionable information to accommodate changes for resolution process of different bug types. For each cell, we import preprocessed sublog generated during the progression of an issue into Disco to discover runtime process model and other statistical information. Disco miner is based on the proven framework of Fuzzy Miner with completely new set of process metrics and modeling strategies12. We select ITS issue ID as CaseID (for process map generation in Disco) to associate all activities pertaining to same issue ID. As a result, visualize the complete life cycle of a bug and control transfer between different systems.

We have a log of 704 and 2271 cases corresponding to Cell 1 (Performance) and Cell 2 (Security) respectively of process cube in Fig. 2. There are total 11 states in the process map where 8 states are for ITS, and 3 for PCR and VCS. Fig. 3a and Fig. 3b depicts process map for issues with Type as Performance and Security respectively. For the sake of simplicity and clarity, only the core transitions are presented in process map because the model with all infrequent transitions is complex and looks like spaghetti. Label on edges represents absolute frequency of transition and value in an activity node is 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. Dark shade is for comparatively higher frequency, thick edge corresponds to more frequent transition. We perform statistical analysis of derived process map for activity frequency, transitions and commonalities.

A. Activity Frequency Analysis

Patch Creation: Patch is submitted to PCR for review in order to resolve an issue. We identify that a significant number of Performance issues, that is, 328 (46.59%) have atleast one patch submitted to PCR while 736 (32.40%) of Security issues have patch(es) to resolve an issue. For both the Types, majority of the cases have 1 or 2 PCR issues in their lifecycle as clearly shown in Fig. 4a. However, there are outliers with more than 5 PCR issues to resolve same ITS issue. From Fig. 3a and 3b, the total number (absolute frequency) of PCR issues created (activity C_Creation) for Type Performance and Security are 741 and 1539 respectively. Out of these PCR issues, 674 (90.95%) and 1473 (95.71%) are committed for Performance and Security respectively. Hence, we can say that comparatively more Performance issues require patches for resolution when in fact, the patches submitted for Security issues are more likely to get committed. Overall the quality of submitted patches is good as the percentage of patches getting committed is quite high.

Closed Status: From Fig. 4b, we notice that there is a significant difference in closed status distribution between two issue types. Interestingly, around 95% of the Security issues are getting resolved as Fixed which is quite high and reflects effective bug resolution. Whereas, only 50% Performance issues are Fixed. A good number of Performance issues are marked as Duplicate or WontFix which means that either the reported issues do not have sufficient details, hence, not reproducible or are not relevant. Very less Security issues are marked as Duplicate or WontFix which highlights the importance given to Security issues. Therefore, efforts should

![Fig. 4: Activity frequency analysis](http://fluxicon.com/disco/files/Disco-Tour.pdf)

---

11https://github.com/Mining-multiple-repos-data/ProcessCubeData
be made to fix more Performance issues and improve the quality of resolution for them.

According to Google Chromium bug resolution guidelines\(^7\), it is recommended that a fix is Verified (means that the fix has been tested and confirmed). However, we notice that very less percentage of issues are verified for their resolution. It is even lesser for Security issues (5\%) as compared to Performance issues (around 18\%) signaling the need to make more efforts towards issue verification.

B. Transition Analysis

Unique Traces: Ordered sequence of activities performed during the lifecycle of a case is known as trace. We observe fairly large number of trace variants where we consider two variants unique even if one transition is different. We have total number of variants as 279 and 549 for Performance and Security type issues respectively. The distribution is skewed, as a result, top 5\% traces depict 50\% and 67\% of cases for Performance and Security issues respectively. We find that for both the types, most common traces have lifecycle limited to ITS only, and not spanning to PCR and VCS. It indicates that the traces spanning across all the 3 IS have more variants with less frequency because majority cases have one while some have multiple PCR issues in their lifecycle. These multiple PCR issue have different ordering of C\_Creation and V\_Commit.

Events in Lifecycle: Total number of events in lifecycle ranges from 3 to as high as 46 and 38 for Performance and Security issues respectively. We observe that the resolution process is quite stable as majority of the cases have less than 10 events with some outliers requiring more activities for resolution. General trace with 3 events can be represented as: $I\_Creation \rightarrow Closed \ Status \rightarrow I\_Closed,$ while the general trace with 4 activities: $I\_Creation \rightarrow I\_Open \rightarrow Closed \ Status \rightarrow I\_Closed$

Control Transfer between 3 IS: As evident from Fig. 3a and 3b, majority of the cases (645 for Performance and 2206 for Security) start their lifecycle from ITS with activity, $I\_Creation.$ For cases with process spanning across all the 3 IS, control transfer between three systems in most of the cases is as $ITS \rightarrow PCR \rightarrow VCS,$ that is, lifecycle starts from ITS. According to Google Chromium bug resolution guidelines\(^13\), it is recommended to first report an issue in ITS followed by patch submission to PCR for resolution thereafter, if approved gets committed to VCS. However, we identify the cases where this practice is not followed and lifecycle begins with patch submission to PCR ($C\_Creation$). 18\% (59 cases) of total Performance issues with resolution spanning 3 IS start their lifecycle from PCR system. Unlike Performance issues, only 8.42\% (62) of Security cases have lifecycle starting from PCR system with delayed issue reporting in ITS to maintain

\(^7\)http://dev.chromium.org/developers/contributing-code
a record of code changes with respect to issue to be addressed.

**Anti-patterns:** We identify undesired, erroneous transitions and cause of their existence in our process model. It is important for a process analyst to improve overall quality by minimizing occurrence of such transitions. We notice from derived runtime process model (refer Fig. 3a and 3b) that \(I_{Closed}\) is end state for comparatively less percentage of cases, that is, 57.30% and 40.82% for Performance and Security respectively. Ideally it should be end state for all the cases. However, we observe outgoing edges from \(I_{Closed}\) towards other states like \(I_{Open}\), \(Fixed\), \(Duplicate\), \(WontFix\), \(C_{Creation}\) and \(C_{Reviewed}\). Potential reasons for outgoing edge to the following states are as follows:

- \(I_{Open}, Fixed, Duplicate, WontFix\): Issue is closed but if the resolution deemed incorrect, it is reopened. Such cases increase maintenance cost, degrade overall user perceived quality of the bug fixing process and leads to unnecessary rework by busy practitioners [14]. We find that there are total 133 and 892 cases with transitions from \(I_{Closed}\) to other states of ITS for Performance and Security issues respectively, which signifies that the issue was reopened and resolved. Interestingly, out of 892 cases of Security issues, 880 transitions are to state \(Fixed\) which means that either additional information becomes available or the root cause is understood later. This leads to fixing of otherwise closed bugs. Also an issue is reopened if a bug regresses. Overall, we can say that more Security issues (around 39.27%) are reopened and preferably \(Fixed\). It is good from the perspective that maximum Security issues are \(Fixed\), at the same time, reopening of many issues indicate inefficiency and delay in resolution process. Therefore, process owner should make efforts to fully understand an issue before marking it closed to minimize chances of reopen.

- \(C_{Creation}, V_{Committed}, C_{Reviewed}\): For Performance and Security issues, there are 133 and 410 direct transitions respectively to PCR and VCS states indicating that the submitted patch review is in progress while the issue gets closed.

**Signature Patterns:** Set of transitions occurring commonly in both the process models and are interesting to study as part of issue resolution lifecycle are marked as signature patterns. From Fig. 3a and 3b, we observe a clear cycle as \(C_{Creation} \rightarrow V_{Committed} \rightarrow C_{Reviewed} \rightarrow C_{Creation}\), which means that for majority of the cases with multiple PCR issues, the PCR issues are created sequentially. It is because if the committed patch is not sufficient to address an issue only then the need for more code changes is realized thereby, followed by new patch report in PCR. For both the processes, we also observe self loop in \(C_{Creation}\) indicating that subsequent patches are submitted to PCR even before the review of previous patch. However, this is happening for comparatively less cases (63 and 72 for Performance and Security issues respectively) as it is not easy to identify the next files to be manipulated without committing already suggested changes.

### V. Time Perspective

Gupta et al. identify specific part within the runtime process map which is relatively time consuming and reduces overall performance of the end-to-end process, that is, bottleneck for stand-alone process [15]. We propose metric to identify bottlenecks using multiple process variants to help process analyst improve the performance. We observe that some transitions are very time consuming in a process irrespective of other variants (General bottleneck) while there exists transitions which are inefficient in some variants when compared with others (that is, Exclusive bottleneck).

**A. Exclusive Bottleneck**

Some transitions are present in all variants. Still mean time for the same transition varies by a factor of 2 or more across the variants. It implies that same activity is performed more efficiently in some variants highlighting the scope of improvement in others. We propose Bottleneck Ratio (\(BNR\)) to identify such instances:

\[
BNR = \frac{|(P_t - P_t')|}{\min(P_t, P_t')}
\]

where \(P_t\) = Mean time for transition \(t\) in process 1, and \(P_t'\) = Mean time for transition \(t\) in process 2

**BNR** measures the extent of difference with respect to minimum time taken for the same transition. If \(BNR > 1\), it means that time taken by a process is atleast double of minimum time for the transition. We suggest use of \(BNR\) to compare two processes while if the number of process variants is more than two then Adjusted Box Plot can be used [16]. Outlier in box plot corresponds to the bottleneck, hence, need for improvement.

Here, we identify bottleneck (using \(BNR\)) present in core transitions of two process variants presented in Fig. 3a and 3b. Some of the examples are listed in Table IV where we find that \(BNR\) ratio is quite high for some of the transitions.

1) As observed in Table IV, identification of \(Duplicate\) and \(WontFix\) issues is very inefficient for Performance issues with \(BNR\) greater than 15. Also from Fig. 4b, percentage of such issues is more for Performance type. Thus, there is clear need to address the cause of delay in determining such issues to improve the quality of Performance issue resolution process.

2) Fig. 3a and 3b reveal that in comparison to Performance issues, Security issues have more open status label in their lifecycle (around 70%), that is, activity \(I_{Open}\) is more frequent in Security issues as

![Adjusted Box Plot](image-url)

**TABLE IV: Transition with BNR where \(P_t\) Corresponds to Performance and \(P_t'\) to Security. Minimum of Two is Represented in Italics. (h: hour, d: day, w: week)**

<table>
<thead>
<tr>
<th>Transition</th>
<th>(P_t)</th>
<th>(P_t')</th>
<th>BNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>(I_{Creation} \rightarrow I_{Open})</td>
<td>17.2 d</td>
<td>4.9 d</td>
<td>2.31</td>
</tr>
<tr>
<td>(I_{Creation} \rightarrow Duplicate)</td>
<td>37.5 d</td>
<td>50.5 h</td>
<td>16.82</td>
</tr>
<tr>
<td>(I_{Creation} \rightarrow WontFix)</td>
<td>24.2 w</td>
<td>8.4 d</td>
<td>19.16</td>
</tr>
<tr>
<td>(V_{Committed} \rightarrow C_{Reviewed})</td>
<td>6.7 d</td>
<td>36.1 d</td>
<td>4.38</td>
</tr>
<tr>
<td>(C_{Reviewed} \rightarrow Fixed)</td>
<td>16.3 d</td>
<td>17.8 w</td>
<td>6.64</td>
</tr>
</tbody>
</table>
compared to Performance issues. Only 49.5% of the Performance issues are labeled with open status label during their lifecycle, even if labeled, the delay is more than Security issues (BNR is 2.51). Henceforth, it highlights the need to emphasize the importance of defined labeling practices in more efficient way for Performance issues.

3) We know that more percentage of Performance issues have patches for resolution. We observe that transitions from VCS to PCR and PCR to ITS is more efficient also as compared to Security issues. Delay to close PCR issue after successful commit is much higher for Security issues with BNR value 4.38. Similarly for marking an issue as Fixed after completion of patch review is taking too much time in case of Security issues (that is, BNR = 6.64).

B. General Bottleneck

Experimental results demonstrate transitions which take lot of time delaying an individual process. For example, following results are few actionable information for the project team:

1) We notice that the mean time to reopen and fix closed issues (\(I_{Closed} \rightarrow Fixed\)) is very high, that is, 29.8 weeks for Security issues. At the same time this transition is present in many cases indeed, it emphasizes the need to minimize issue reopen.

2) In case of Security issues, it takes mean time of around 3 months to directly fix an issue that is, transition from \(I_{Creation} \rightarrow Fixed\). On the other hand, sequence of transitions as \(I_{Creation} \rightarrow I_{Open} \rightarrow Fixed\) takes mean time of around 2 months which indicates that proper assignment of issue before jumping to the resolution reduces overall delay.

3) Reopening of Performance issues, that is, \(I_{Closed} \rightarrow I_{Open}\) takes mean time of 54.9 days which is undesirable and degrades the overall user perceived quality of the bug fixing process.

4) Resolution of an issue as WontFix or Invalid is taking significantly high mean time of around 21 and 17 weeks respectively for Performance issues. It indicates inefficiency in identifying the issues which are not worth fixing because of several reasons like insufficient information.

5) Control transfer from ITS state \(I_{Open}\) to PCR state \(I_{Creation}\) is taking 11.3 and 15 days for Security and Performance issues respectively. It indicates scope of improvement in making transition smooth between different IS.

VI. Conclusion

We model process cube with 9 dimensions for defect resolution process. We handle sparsity for selected dimension \(D_{col}\) by meaningful clustering and hierarchies. We perform a set of OLAP operations (slice, dice and roll-up) to eventually compare Performance and Security issues. We derive two sublogs (each corresponding to a cell) from EB by materialization step. We obtain process model with 11 states which helps to visualize differences. Empirical analysis reveals interesting patterns like more Performance issues have patches, high percentage of Security issues are Fixed. We identify high number of unique traces for both Types with majority cases having 3 or 4 events in the lifecycle. We find exceptional control transfer in 18% of Performance issues. We observe frequent reopen cases (anti-patterns) for Security issues and presence of code review cycles (signature patterns). We identify Exclusive bottlenecks using BNR and also General bottlenecks delaying individual process.

ACKNOWLEDGEMENT

The presented research work is supported by Prime Minister’s fellowship awarded to the first author. I thank my industry mentor, Dr. Srinivas Padmanabhuni for valuable inputs. I acknowledge SERB, CII and Infosys Limited for their support.

REFERENCES


