Volvo IT Incident Handling

Process Mining Analysis

Kaleabe Seifu Desta

July 08,2025

Contents

Volo IT Incident Handling Process Mining Analysis
Description of the Case Study

1 Introduction

The Volvo IT Incident Handling case study involves analyzing the event log of the incident management process within Volvo IT Belgium, captured in the VINST system. Volvo IT provides IT services according to terms and conditions regulated in Service Level Agreements (SLAs). The VINST system is used by Volvo IT to support incidents and problems reported by the IT service users. The primary goal of the Incident Handling process is to restore normal service operation as quickly as possible and by that ensure that the best possible levels of service quality and availability are maintained".

In particular, processes are organized around three support lines. Each incident is preferred to be solved by Support Teams (STs) working in the 1st line. The incident is forwarded to the other lines only when it cannot be handled in the 1st line. Volvo IT has pointed particularly focuses on identifying inefficiencies, particularly ping-pong behavior **(**this aspect refers to the unwanted pattern in interaction among STs, when a request is repeatedly bounced from one ST to another**)** and has asked for in-depth analysis and assessing process conformance to a reference model to improve operational efficiency. [1]

2 Methodology

The method used to analyze and exploit data about business processes is Knowledge Uplift Trail (KUT). Through this method, we can go from raw data Gaining clear, data-driven insights into these areas is crucial for optimising the incident management process, reducing resolution times and ultimately improving IT service delivery.

---

[1] *Analysis of the Volvo IT Incident and Problem Handling Processes using Process Mining and Social Network Analysis* https://www.ceur-ws.org/Vol-1052/paper10.pdf *Page 2*

3 Case study

The Incident Handling event log captures 65.533 events generated during the execution of 7.554 process instances. There are 13 activity types in the Incident Handling log with main attributes of these cases are:

1. **1.** case:concept:name: Unique process instance ID (one per incident).
2. **c**oncept:name: Activity name (status or sub-status change in VINST system).
3. time:timestamp: Timestamp when the activity occurred.
4. org:group: The team/department responsible for the activity. (in preprocessing it is split into support:line and support:team)
5. lifecycle:transition: Lifecycle stage of the activity (used in preprocessing to get all the activities in the case study)
6. organization involved: Organization participating in the activity aka departments for this analysis

4 Data Preprocessing

These preprocessing steps ensure that the event log is clean, standardized, and perfectly structured for detailed process mining.

4.1 Data modifications

➢ Merge original concept:name and lifecycle:transition into a new concept:name (e.g., “Accepted – In progress”, “Queued/Awaiting Assignment”) which makes the number of activity types in the incident handling log 13.

```
#Merge lifecycle:transition and concept:name to get all activities
df_log['concept:name'] = (
    df_log['concept:name'].astype(str) + "/" +
    df_log['lifecycle:transition'].astype(str)

)
activities = df_log['concept:name'].value_counts()
print("\nActivity types in the Incident Handling logp':")
print(activities)
#relevant_coluns to keep
relevant_columns = [
    'case:concept:name', 'concept:name', 'time:timestamp',
    'org:group', 'org:role', 'organization involved'
]
df_log = df_log[relevant_columns]
✓ 0.0s
```

```
Activity types in the Incident Handling logp':
concept:name
Accepted/In Progress             30239
Queued/Awaiting Assignment       11544
Completed/Resolved                6115
Completed/Closed                  5716
Accepted/Wait - User              4217
Accepted/Assigned                 3221
Completed/In Call                 2035
Accepted/Wait                     1533
Accepted/Wait - Implementation     493
Accepted/Wait - Vendor             313
Accepted/Wait - Customer           101
Unmatched/Unmatched                  5
Completed/Cancelled                  1
Name: count, dtype: int64
```

➢ Split org:group into support team(e.g., “G96”) and support line number (e.g., “1st”, “2nd”, “3rd”, “2nd-3rd”).

```
df_log = df_log.copy()
df_log['support:level'] = df_log['org:group'].str.extract(r'(1st|2nd|3rd|2nd-3rd)')
# Clean support team by removing support line number
df_log['support:team'] = df_log['org:group'].str.replace(r'(1st|2nd|3rd|2nd-3rd)', '', regex=True).str.strip()

# Assign '1st' to missing support line number
df_log['support:level'] = df_log['support:level'].fillna('1st')

# Validate support line number
valid_support_lines = ['1st', '2nd', '3rd', '2nd-3rd']
invalid_support_lines = df_log[~df_log['support:level'].isin(valid_support_lines)]['support:level'].unique()
if len(invalid_support_lines) > 0:
    print(f"Warning: Invalid support line number values found: {invalid_support_lines}. Replacing with '1st'.")
    df_log.loc[~df_log['support:level'].isin(valid_support_lines), 'support:level'] = '1st'

# Exclude events missing Org line
if df_log['organization involved'].isna().any():
    print(f"Warning: {df_log['organization involved'].isna().sum()} events missing Org line. Dropping affected rows.")
    df_log = df_log.dropna(subset=['organization involved'])
```

The 1st line number was assumed for activities for which no line number was mentioned.

4.2 Time equal to zero

The next step taken is to find and remove all cases with duration equal to 0.

```
# Calculate the duration of each case
case_durations = filtered_log.groupby('case:concept:name').agg(Duration=('time:timestamp', lambda x: (x.max() - x.min()).days))
filter = case_durations[case_durations['Duration'] > 0]
filtered_log = filtered_log[filtered_log['case:concept:name'].isin(filter.index)]
# Print the number of unique active cases with a duration greater than zero
print('Number of active cases: {}'.format(len(filtered_log['case:concept:name'].unique().tolist())))
✓ 0.2s

Number of active cases: 5337
```

After seeing the lowest time it’s confirmed that there are no cases with time equal to zero.

5 Data Filtering

After preprocessing the data, it’s mandatory to remove and filter the log to get the information that are useful to the case study for ping-pong and conformance analysis.

5.1 Remove cases that are very short

Filter the event log to remove cases that have two or fewer events. It first counts the number of events for each unique case, then identifies cases with more than two events, and finally creates a new filtered_log DataFrame containing only the events belonging to these longer cases.

```
# Remove cases with <=2 events
case_counts = df_log['case:concept:name'].value_counts()
filtered_log = df_log[df_log['case:concept:name'].isin(case_counts[case_counts > 2].index)]

✓ 0.0s
```

5.2 Remove outlier duration

Calculate the total duration for each case in the event log, identifies and removes cases whose durations are statistical outliers (specifically, those exceeding three standard deviations above the mean duration). This is done to ensure that the analysis focuses on typical process behaviors and is not skewed by extremely long or erroneous cases, thereby improving the reliability of subsequent process mining results.

```python
# Compute case durations in hours
filtered_log = filtered_log.copy()
filtered_log['time:timestamp'] = pd.to_datetime(filtered_log['time:timestamp'])

case_durations = filtered_log.groupby('case:concept:name')['time:timestamp'].apply(
    lambda x: (x.max() - x.min()).total_seconds() / 3600
)

# Compute mean and standard deviation
mean_duration = case_durations.mean()
std_duration = case_durations.std()
threshold = mean_duration + 3 * std_duration  # 3σ rule

print(f"Mean Duration: {mean_duration:.2f} hours")
print(f"Standard Deviation: {std_duration:.2f} hours")
print(f"Outlier Threshold (Mean + 3σ): {threshold:.2f} hours")

# Filter out cases above threshold
valid_cases = case_durations[case_durations <= threshold].index
filtered_df = filtered_log[filtered_log['case:concept:name'].isin(valid_cases)]

print(f"Original Cases: {filtered_log['case:concept:name'].nunique()}")
print(f"Filtered Cases (no outliers): {filtered_df['case:concept:name'].nunique()}")
```

✓ 0.1s

```
Mean Duration: 409.01 hours
Standard Deviation: 785.79 hours
Outlier Threshold (Mean + 3σ): 2766.37 hours
Original Cases: 5337
Filtered Cases (no outliers): 5255
```

5.3 Remove Infrequent activities

After checking the activity counts and calculating the percentage of the occurrence of each activities then filters the event log to keep only activities that occur frequently (at least 1% of total events) or are explicitly kept as crucial "end activities," like "Completed/In Call" thereby simplifying the process model by removing noise and focusing on the most relevant and significant process steps.

```python
# Calculate activity frequencies
activity_counts = filtered_df['concept:name'].value_counts(normalize=True) * 100
# Identify frequent activities (>=1% of total events)
frequent_activities = activity_counts[activity_counts >= 1].index.tolist()

# KEEP Completed/In Call for its one of the end activites
keep_rare_activitie = [
    'Completed/In Call',
]
# Combine frequent activities and rare ones to keep
final_activities = set(frequent_activities).union(set(keep_rare_activitie))

# Filter the log
df_log_cleaned = filtered_df[filtered_df['concept:name'].isin(final_activities)]

# Summary of cleaned log
print("Original activities:", filtered_df['concept:name'].nunique())
print("Cleaned activities:", df_log_cleaned['concept:name'].nunique())
print("Remaining events:", len(df_log_cleaned))
print("\nActivity counts in cleaned log:")
print(df_log_cleaned['concept:name'].value_counts())
```

	Activity	Count	Percentage
0	Accepted/In Progress	24505	44.111823
1	Queued/Awaiting Assignment	10694	19.250432
2	Completed/Resolved	5612	10.102247
3	Completed/Closed	5243	9.438004
4	Accepted/Wait – User	3986	7.175259
5	Accepted/Assigned	3057	5.502952
6	Accepted/Wait	1406	2.530962
7	Accepted/Wait – Implementation	452	0.813652
8	Accepted/Wait – Vendor	299	0.538234
9	Completed/In Call	195	0.351022
10	Accepted/Wait – Customer	98	0.176411
11	Unmatched/Unmatched	5	0.009001

5.4 Filter start and end activities

The majority of Handle Incident process instances starts with an \Accept/In Progress" (84.35%) or “Queued/Awaiting Assignment" (15.3%) activity. Four activities never start the Incident Handling process: \Assigned/Wait-customer", ”Unmatched/Unmatched", “Completed/Closed", ”Completed/Cancelled". The majority of process instances is finished by “Completed/Closed" (73.77%) while process instances finished by “Completed/In Call", “Completed/Resolved" and “Completed/Cancelled" accounts for 26.1% of all the process instances. Eight process instances (0.1%) are still running. [2]

```
# Allowed start activities
allowed_start_activities = [
    "Accepted/In Progress",
    "Queued/Awaiting Assignment"
]
# Allowed end activities
allowed_end_activities = [
    "Completed/Closed",
    "Completed/Resolved",
    "Completed/In Call"
]
# First filter start activities
case_start_activities = (
    df_log_cleaned.sort_values("time:timestamp")
    .groupby("case:concept:name")
    .head(1)
)[['case:concept:name', 'concept:name']]
valid_start_cases = case_start_activities[
    case_start_activities['concept:name'].isin(allowed_start_activities)
]['case:concept:name']

# Then filter end activities
case_end_activities = (
    df_log_cleaned.sort_values("time:timestamp")
    .groupby("case:concept:name")
    .tail(1)
)[['case:concept:name', 'concept:name']]
valid_end_cases = case_end_activities[
    case_end_activities['concept:name'].isin(allowed_end_activities)
]['case:concept:name']

# Convert Series of case IDs to sets
valid_start_set = set(valid_start_cases)
valid_end_set = set(valid_end_cases)

# Find intersection (common cases)
valid_cases = valid_start_set.intersection(valid_end_set)

# Keep only cases valid in both filters
filtered_log = df_log_cleaned[df_log_cleaned['case:concept:name'].isin(valid_cases)].copy()

print(f"Remaining Cases After Start+End Activity Filter: {filtered_log['case:concept:name'].nunique()}")
```

5.5 Dataset After Cleaning

This section outlines the attributes of the dataset after data cleaning. The cleaned dataset includes the following start and end activities :

- ***Start activities:*** 'Accepted/In Progress': 4127, 'Queued/Awaiting Assignment': 1105
- ***End activities:*** 'Completed/Closed': 5092, 'Completed/In Call': 69, 'Completed/Resolved': 71

[2] Analysis of the Volvo IT Incident and Problem Handling Processes using Process Mining and Social Network Analysis, *op. cit.*, p. 3.

Knowledge Uplift Trail: Process Mining Analysis Summary

Phase	Step	Actions	Key Insight
1. Data Preparation	Load the event log and preprocess it.	● Convert to pm4py event log ● Merge lifecycle:transition and concept:name to get all activities and select relevant columns to keep. ● Split org:group into support:team and support:level number extract support level number (e.g., "1st", "2nd", "3rd", "2nd-3rd")	These preprocessing steps ensure that the event log is clean, standardized, and perfectly structured for detailed process mining.
2. Filtering	Applied filters to focus on relevant cases for ping-pong and conformance analysis.	● Filtered cases that are very short(with case count <=2 events) ● Calculate case durations, identifies outliers using a 3-sigma rule (mean + 3 * standard deviation), and then filters out these outlier cases to refine the dataset ● Identify frequent activities (>=1% of total events) and remove the infrequent ones but keep those activities that are of	Isolated cases likely to exhibit ping-pong behavior and ensured data quality, focusing on complex, multi-team incidents.

		the end activities. ● Filter start and end activities ● Group events in to variants and keep choose the top 30 variants considering pareto distribution for further analysis	
3. Segmentation	Segmented the log by support:level, organization involved	● Created sub-logs for each unique value in the respective attributes support:level units like "1st", "2nd", "3rd", "2nd-3rd"). ● Identified top 3 organization involved units (C, A2, G4) by event count for focused analysis.	Enabled analysis of process variations across different dimensions, with C (1,865 traces) and A2 (675 traces) showing high involvement.
4. Ping-Pong Behavior Detection	Detected ping-pong behavior (transfers between org:group units).	● Defined ping-pong as sequences where three consecutive events involve different org:group units (A to B to A) for activities like “Accepted – In Progress”, “Accepted Assigned”, or “Queued – Awaiting Assignment”. ● Excluded intra-team reassignments. ● Summarized frequency, duration, and most involved org:group units. ● Visualized using: Bar Chart.	Identified frequent ping-pong between C and A2, contributing to process inefficiencies.

5.Conformance Checking	Assessed how well traces for top 3 org:group units (C, A2, G4) conform to a reference Petri net.	• Discovered a reference Petri net using Inductive Miner (pm4py.algo.discovery.inductive, variant IM). • Performed token-based replay (pm4py.algo.conformance.tokenreplay) to compute: • Visualized DFGs for each organization and the reference Petri net.	Extremely low conformance indicates significant deviations, likely due to ping-pong transfers, non-standard handovers, or specialized roles (e.g., G4).

6 Data Segmentation

The segmentation phase divides the preprocessed and filtered Volvo IT Incident Handling event log into sub-logs based on key attributes: support:level (e.g., 1st, 2nd, 3rd Line), *organization involved* (organizational units like C, A2, G4). This enables targeted analysis of process variations across different dimensions. The top 3 *organization involved*/ departments units (C, A2, G4) were identified by event count for focused analysis, revealing high involvement of C (1,865 traces) and A2 (675 traces), with G4 (149 traces) indicating a specialized role. Segmentation supports ping-pong detection, variant analysis, and conformance checking by isolating relevant subsets of the log.

```
support_line_logs = {}
for lines in filtered_df['support:level'].unique():
    lines_log = filtered_df[filtered_df['support:level'] == lines]
    support_line_logs[lines] = lines_log
    print(f"Support Line {lines}: {lines_log['case:concept:name'].nunique()} cases")
```

✓ 0.0s

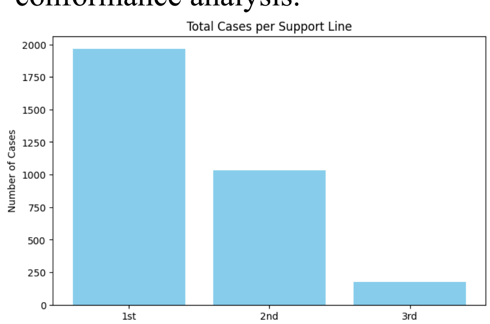
```
Support Line 2nd: 1031 cases
Support Line 3rd: 177 cases
Support Line 1st: 1965 cases
```

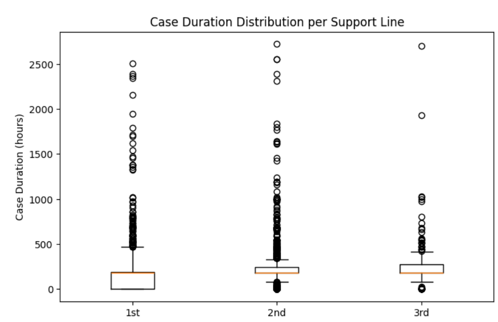
```
# Count cases per organization
org_case_counts = filtered_df.groupby('organization involved')['case:concept:name'].nunique()
# Get top 3 organizations by case count
top_3_orgs = org_case_counts.sort_values(ascending=False).head(3).index.tolist()
# Segment logs for top 3 organizations
org_line_logs = {}
for org in top_3_orgs:
    org_log = filtered_df[filtered_df['organization involved'] == org]
    org_line_logs[org] = org_log
    print(f"Organization {org}: {org_log['case:concept:name'].nunique()} cases")
```

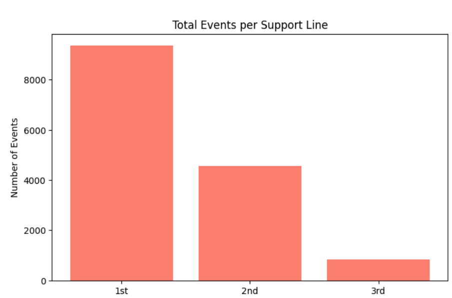
✓ 0.0s

```
Organization Org line C: 1865 cases
Organization Org line A2: 675 cases
Organization Org line G4: 149 cases
```

After the segmentation we iterate through different support lines, calculate key statistics for each (total cases, total events, mean/median case duration), and then visualize these metrics using bar charts to compare total cases and events, and a boxplot to show the distribution of case durations across support lines. We keep the organization/departments segment for the cross-Department conformance analysis.







7 'Ping-pong' behaviour detection

This analysis aims to identify "ping-pong behavior" in support processes, defined as incidents repeatedly bounced between different support teams.[3] The methodology focuses on specific criteria: Defined as transfers between different org:group units (e.g., N36 → N51) for activities like "Accepted – In Progress", "Accepted – Assigned", or "Queued – Awaiting Assignment", with ≥6 events and >1 '*org:group*'. Intra-team reassignments are excluded.

```python
def detect_ping_pong(df):
    ping_pong_transitions = []
    # Define activities to consider for ping-pong behavior
    ping_pong_activities = [
        'Accepted/In Progress', 'Accepted/Assigned', 'Queued/Awaiting Assignment'
    ]

    # Sort log by case and timestamp
    df_sorted = df.sort_values(['case:concept:name', 'time:timestamp'])

    # Group by case
    for case_id, group in df_sorted.groupby('case:concept:name'):
        group = group.reset_index(drop=True)
        activities = group['concept:name'].str.split('_').str[0]  # Remove lifecycle suffix
        org_groups = group['org:group']
        timestamps = group['time:timestamp']

        for i in range(len(group) - 2):
            current_activity = activities[i]
            next_activity = activities[i+1]
            return_activity = activities[i+2]

            # Check ping-pong pattern
            if (current_activity in ping_pong_activities and
                next_activity in ping_pong_activities and
                return_activity in ping_pong_activities and
                org_groups.iloc[i] != org_groups.iloc[i+1] and
                org_groups.iloc[i] == org_groups.iloc[i+2]):

                group_pair = (org_groups.iloc[i], org_groups.iloc[i+1])
                duration = (timestamps.iloc[i+2] - timestamps.iloc[i]).total_seconds() / 3600

                ping_pong_transitions.append({
                    'case_id': case_id,
                    'group_pair': group_pair,
                    'activity_sequence': f"{current_activity} -> {next_activity} -> {return_activity}",
                    'duration': duration,
                    'start_time': timestamps.iloc[i]
                })

    return pd.DataFrame(ping_pong_transitions)

# Apply to your filtered DataFrame
ping_pong_df = detect_ping_pong(filtered_df)

# Preview results
print(ping_pong_df.head())
```
✓ 0.4s

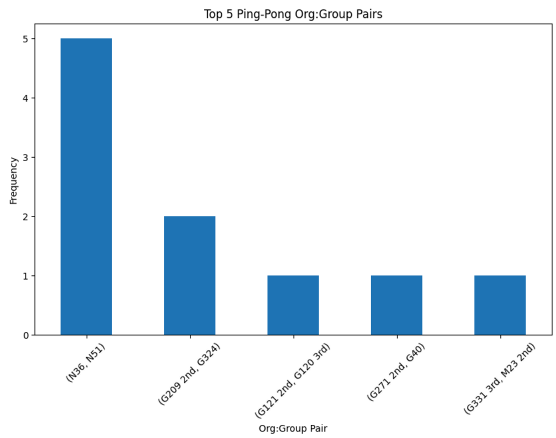
Count occurrences of each activity sequence in the ping-pong behavior and the duration for those activities:

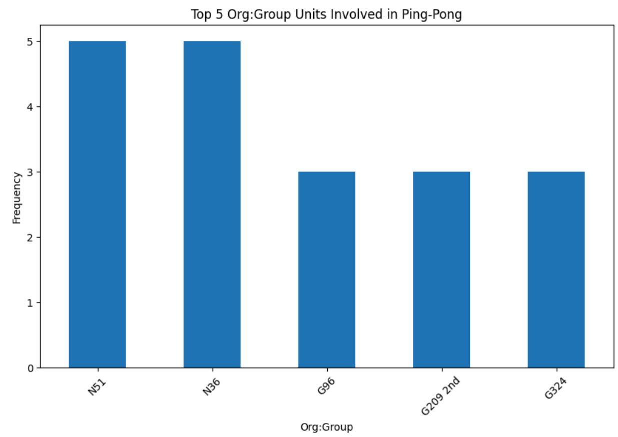
```
                                                    activity_sequence  count  average_duration
      Accepted/In Progress -> Queued/Awaiting Assignment -> Accepted/In Progress     24         43.471863
Queued/Awaiting Assignment -> Accepted/In Progress -> Queued/Awaiting Assignment      4         11.855000
Accepted/In Progress -> Queued/Awaiting Assignment -> Queued/Awaiting Assignment      1         92.241944
```

[3] John Hansen, ChangeGroup Partner International Business Process Intelligence 2013 Competition https://www.ceur-ws.org/Vol-1052/paper6.pdf page 10

Visualizing ping-pong org:group pairs

Identifying the most frequent organizational group(Support teams) pairs involved, and calculating overall ping-pong frequency and average duration, now we visualise the top pairs and most involved groups using bar charts.






```
Most common ping-pong org:group pairs:
group_pair
(N36, N51)            5
(G209 2nd, G324)      2
(G121 2nd, G120 3rd)  1
(G271 2nd, G40)       1
(G331 3rd, M23 2nd)   1
Name: count, dtype: int64

Ping-pong frequency: 29
Average ping-pong duration (hours): 40.792643678160914

Most involved org:group units in ping-pong behavior:
group_pair
N51         5
N36         5
G96         3
G209 2nd    3
G324        3
Name: count, dtype: int64
```

8 Cross-Department Conformance

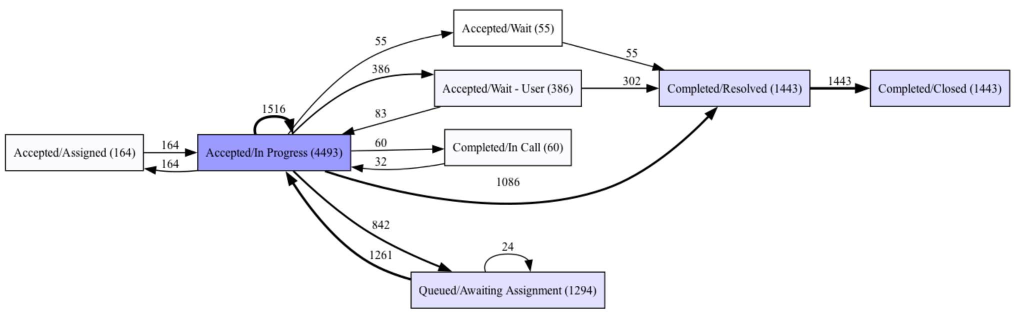
When a reference model is available for the process, it is possible to check the process conformance of the execution of the process. As the reference model of the incident VISITS Support Process is not available, it is not possible to conduct a process conformance check on the reference model. However, it is possible to compare the process execution models for the two IT organizations Org line A2 and Org line C.[4]

[4] Emmy Dudok, Peter van den Brand: Mining an Incident Management Process
https://www.ceur-ws.org/Vol-1052/paper4.pdf page 19

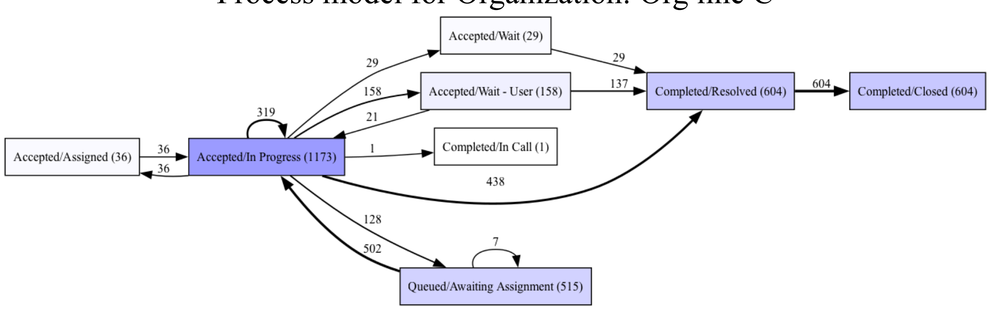
- The **inductive miner** This model provides a complete and generalized view of possible scenarios during process execution, capturing a wide range of potential situations and offering a robust overview of the entire process. So For each of these selected organizations, it then extracts their specific event log and applies the Directly-Follows Graph (DFG) discovery algorithm to generate a visual representation of their actual process flow.

```python
from pm4py.algo.discovery.dfg import algorithm as dfg_discovery

# Extract DFGs
dfg_a2 = dfg_discovery.apply(log_a2)
dfg_c = dfg_discovery.apply(log_c)

print("DFG A2:", dfg_a2)
print("DFG C:", dfg_c)
```

✓ 0.0s



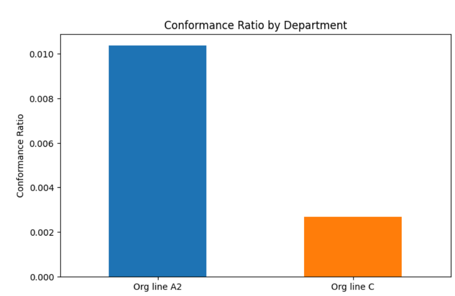
Process model for Organization: Org line C



Process model for Organization: Org line A2

- Then compare the Directly-Follows Graphs (DFGs) of "Org A2" and "Org C" to identify common process steps, as well as unique transitions specific to each organization, providing insights into their shared and distinct process behaviors.

```python
# Convert DFGs to sets for comparison
dfg_a2_set = set(dfg_a2.items())
dfg_c_set = set(dfg_c.items())

common_edges = dfg_a2_set.intersection(dfg_c_set)
a2_unique = dfg_a2_set - dfg_c_set
c_unique = dfg_c_set - dfg_a2_set

print("Common activities/transitions:", common_edges)
print("Unique to Org A2:", a2_unique)
print("Unique to Org C:", c_unique)
```



Key findings of the conformance checking

Common Trace Variants Between Org A2 and Org C:

- ('Accepted/In Progress', 'Accepted/In Progress', 'Completed/Resolved', 'Completed/Closed')
- ('Queued/Awaiting Assignment', 'Accepted/In Progress', 'Completed/Resolved', 'Completed/Closed')
- ('Accepted/In Progress', 'Accepted/In Progress', 'Queued/Awaiting Assignment', 'Accepted/In Progress', 'Completed/Resolved', 'Completed/Closed')

These three variants indicate **shared process behaviors** between the two departments:

- **Repeated work steps** (*Accepted/In Progress* twice)
- **Reassignment loops**
- **Standard resolve-close paths**

Despite these shared variants, the **low conformance ratios** show that there are many **nonconforming variants** unique to each org — especially involving:

- Wait states
- Truncated flows
- Unusual step sequences

Project Results Conclusion

Summary of Findings

The analysis of the Volvo IT Incident Handling event log revealed significant insights into process inefficiencies, particularly **ping-pong behavior** and low conformance to the standard process model. Below is a summary of the key findings and potential improvements to address them, aligning with Volvo IT Belgium's organizational goals of reducing inefficiencies.

Key Findings

1. **Ping-Pong Behavior**: Frequent transfers of incidents between org:group units (e.g., N36 → N51) were identified, particularly for activities like "Accepted – In Progress", "Accepted – Assigned", and "Queued – Awaiting Assignment" in cases with ≥6 events and >1 org:group. Ping-pong transfers introduce redundant steps, increasing resolution times.

2. **Low Conformance**: The organizations differ mostly in the number of incidents handled in this particular period. Furthermore, the most frequent paths differ in that Org line C handles most cases within first line support teams, whereas Org line A2 does not.

9 Proposed improvement

The analysis revealed significant inefficiencies in the Volvo IT Incident Handling process, driven by frequent ping-pong behavior , high process variability. These issues contribute to delays, and inconsistent processes. We suggested potential improvements:

- Implement an automated ticket routing system using rule-based algorithms or machine learning to assign incidents to the appropriate support team based on Impact, Org line, or keywords in the incident description. This reduces manual assignment delays and ensures incidents reach the correct team faster.
- Develop a standardized incident management process model based on the reference DFG from the full log. Enforce this model through VINST system configurations and regular conformance checks using process mining tools.
- Establish clear handover protocols, including mandatory fields in the VINST system (e.g., reason for transfer, expected action) when an incident is reassigned. This reduces unnecessary back-and-forth by ensuring the receiving team has sufficient context.
- Reorder the process to prioritize immediate assignment for high-impact incidents. For example, bypass the "Queued" status for incidents with Impact = "High" by routing them directly to a dedicated team or escalation queue.

will enhance process efficiency, reduce ping-pong, and improve SLA compliance, aligning with Volvo IT Belgium's organizational goals.

10 Code Reference

Additional material can be found at the following link:
https://github.com/seifukaleab/BIS_Project_2025