Process mining of Volvo IT incident management

Antoin Sader

Business Information Systems
Prof. Paolo Ceravolo
a.a. 2024 - 2025

Description of the case study

In this report, we will do a process mining analysis for Incident Management log from Volvo IT Belgium.
The analysis will be focused on 4 key areas:

1. Event log filtering and preparation
2. “Ping pong” behavior detection
3. Cross-department process conformance comparison
4. Improvement suggestions

We use PM4PY library to read .xes file and convert it to Pandas DataFrame for easy data illustration and manipulation, we save dataframes after each step in .pkl files for easy retrieval.

Gituhb repo: https://github.com/antoinsader/process_mining

1- Filtering and preparation:

1-1- Event log filtering and preparation:

1-1-1- Remove incomplete or trivial cases:

A. Filter out the incident cases that are incomplete (does not have **Completed** as the last concept_name) in their cases (8 cases).
B. Filter out cases with very short or trivial sequences (less than 3 events) as may represent auto-closed tickets that do not follow normal process flow (Accepted or queued then Completed). (11 cases)
C. Filter out cases where the start case is Completed as they are abnormal (4 cases)

```
# filter out cases that does not end with Completed
df = df.sort_values(["case_seq_num", "timestamp"])
df = df.groupby("case_seq_num").filter(lambda trace: trace["concept_name"].iloc[-1] == "Completed").reset_index(drop=True)

# Filter out cases with very short or trivial sequences (>2 events)
df = df.groupby("case_seq_num").filter(lambda x: len(x) > 2)



# Filter out cases that they don't start with 'Accepted' or 'Queued'. 4 cases
df_sorted = df.sort_values(["case_seq_num", "timestamp"])
first_events = df_sorted.groupby("case_seq_num").first().reset_index()
valid_case_ids = first_events[first_events["concept_name"].isin(["Accepted", "Queued"])]["case_seq_num"]
df = df_sorted[df_sorted["case_seq_num"].isin(valid_case_ids)]
```

Code snippet showing remove incomplete or trivial cases

After this step, every remaining case has a proper start (Accepted, Queued) state and end (Completed) state and at least one event representing meaningful processing.

1-1-2- Filter out outlier cases by duration:

- Remove extremely long cases, we can remove for example cases where the duration in days (difference between first timestamp and last one) is more than 365 days (we have 8 cases)
- What we preferred to do is filter out cases beyond the 99th percentile of duration (more than 116 days) which are 76 cases from 7385 cases)

```
# Filter out outlier cases by duration
df.loc[:, "timestamp"] = pd.to_datetime(df["timestamp"])
df = df.sort_values(["case_seq_num", "timestamp"]) #sorting other time to make sure
case_durations = df.groupby("case_seq_num")["timestamp"].agg(["first", "last"]).reset_index()
case_durations["duration_days"] = (case_durations["last"] - case_durations["first"]).dt.total_seconds() / (24*60*60)
case_durations = case_durations.sort_values(["duration_days"])
duration_99th = case_durations["duration_days"].quantile(0.99)
valid_case_ids = case_durations[case_durations["duration_days"] < duration_99th ]["case_seq_num"]
df =df[df["case_seq_num"].isin(valid_case_ids)]

```

Code snippet showing the step of filter out outlier cases by duration

1-1-3- Filter noise events:

We have in our logs ***"concept:name"*** which having status of the event ("accepted", "completed", "queued", "unmatched"), if we inspect the frequencies of those, we will notice that we have "unmatched" in only 5 lines which can consider noise and we can filter out. (Dropped 5 events)

```
concept_name
Accepted      39062
Completed     13684
Queued        11144
Unmatched         5
Name: count, dtype: int64
```

Also, we have the attribute "lifecycle:transition" which having ("In progress", "awaiting assignment", etc.). And we can notice from the picture below showing proportions of each one that ["Wait - Implementation", "Wait- vendor", "wait -customer", "cancelled"] are being in less than 1%. Thus, we can consider those as noise and filter them out. (Dropped 79 cases)

```
lifecycle_transition
In Progress                0.461731
Awaiting Assignment        0.174425
Resolved                   0.094115
Closed                     0.088261
Wait - User                0.063860
Assigned                   0.049053
In Call                    0.031789
Wait                       0.023008
Wait - Implementation      0.007356
Wait - Vendor              0.004836
Wait - Customer            0.001550
Cancelled                  0.000016
Name: proportion, dtype: float64
```

```
#filter out noise
concept_name_proportions = df["concept_name"].value_counts(normalize=True)
valid_concepts = concept_name_proportions[concept_name_proportions >= 0.01].index
df = df[df["concept_name"].isin(valid_concepts)]

lctransitions_proportions = df["lifecycle_transition"].value_counts(normalize=True)
valid_lstransitions = lctransitions_proportions[lctransitions_proportions >= 0.01].index
df = df[df["lifecycle_transition"].isin(valid_lstransitions)]

```

Code to filter out noise (removing concept names and lifecycle transitions that are occurred less than 1% of the rows)

1-2- Log segmentation by different support lines/departments:

From the log, we can observe that the organization is divided into multiple lines and departments. The event log provides an attribute **"organization involved"** which has 25 organization lines which are labeled for example "org line c", "org line a2".
We can observe from the log that "Org line C" and "Org line A2", "Org line B" handled most of the incidents (65%, 18%, 7% respectively), so we can consider the logs where those organizations are involved are our points of interest.

Therefore, we can use the organization involved attribute as our attribute to segment the log based on the department responsible for each ticket.

```
org_involved
Org line C      0.653134
Org line A2     0.187871
Org line B      0.071226
Other           0.036090
Org line G4     0.013381
Org line V2     0.009437
Org line V11    0.006808
Org line V7n    0.006119
Org line G1     0.003365
Org line G2     0.002911
Org line V5     0.002301
Org line E      0.001753
Org line F      0.000892
Org line V8     0.000814
Org line V7     0.000736
Org line V10    0.000595
Org line V3     0.000454
Org line D      0.000438
Org line H      0.000438
Org line V9     0.000407
Org line V1     0.000344
Org line G3     0.000250
Org line I      0.000157
Org line V      0.000047
Org line V4     0.000031
Name: proportion, dtype: float64
```

Output of code showing org_involved countings

```
66    c_logs = df[df["org_involved"] == "Org line C"]
67    a2_logs = df[df["org_involved"] == "Org line A2"]
```

Code snippet showing how to segment based on organization involved, and creating 2 dataframes for org C and org A2 which are responsible for most of the events

On the other hand, to segment the log based on different support lines, we can use the attribute "Org:role" which has values like "A2_1", "V3_2", etc.

2- Ping-Pong behavior detection:

"Ping-Pong" behavior refers to the tickets being transferred repeatedly back and forth between teams without resolution, which is undesirable scenario that indicates misrouting or unclear responsibility.

In our context, ping-pong behavior is defined as an incident starting with support team A then being transferred to team B (PING) and later reassigned back to team A (PONG).
This can happen multiple times, causing delays and frustration.
We analyze the log to:

A- Identify most common pairs of teams involved in “ping pong” behavior
B- Quantify the frequency and duration of these handovers

2-1- Identify ping-pong cases and then the pairs of teams involved:

To identify the ping-pong behavior, we group by the dataframe by the attribute “case_seq_num” representing the case-id, then for each trace, we discover ping-pong behavior by dropping consecutive duplicates “org-groups” (e.g. if we have A – B – B – C – A , it will be converted into A – B – C – A), then we loop for every org of the trace, we mark it as ping and we look if the org has been appeared again, then the pong org is the one exactly before the appearance of the second time of the ping item.
We register the duration days starting which is the difference between the timestamps of the two times the org has been appeared, and register the pair of (ping_org, pong_org).
After, we are creating a new dataframe that has the columns: “case_seq_num”, “has_ping_pong”, “ping_pong_pairs” (including the ping item, pong item and the duration in days).

```python
def get_ping_pong(trace):
    orgs = trace["org_group"].tolist()
    timestamps = trace["timestamp"].tolist()
    orgs = [(x, timestamps[i]) for i, x in enumerate(orgs) if i ==0 or x != orgs[i-1] ]
    pairs = []
    n = len(orgs)
    for ping_idx in range(n):
        ping_item = orgs[ping_idx][0]
        ping_timestamp = orgs[ping_idx][1]
        if ping_item in [x[0] for x in orgs[ping_idx:]]:
            for pong_idx in range(ping_idx+1, n):
                ping_again = orgs[pong_idx][0]
                if ping_item == ping_again:
                    duration_days = (orgs[pong_idx][1] - ping_timestamp).total_seconds() / (24*60*60)
                    pairs.append(((ping_item, orgs[pong_idx - 1][0]), (duration_days)))
                    break
    return pairs

def ping_pong_trace_serie(trace):
    pairs = get_ping_pong(trace)
    return pd.Series({
        "case_seq_num": trace.name,
        "has_ping_pong": len(pairs) > 0,
        "ping_pong_pairs": pairs
    })


df = pd.read_pickle("./data/1_log_clean.pkl")
ping_pong_df = df.groupby("case_seq_num").apply(ping_pong_trace_serie).reset_index(drop=True)

num_ping_pong = ping_pong_df["has_ping_pong"].sum()
print(f"Number of traces with ping-pong: {num_ping_pong}")
```

Code shows how to detect ping-pong behavior, grouping logs by case_seq_num then finding ping-pong in each trace

We are considering the trace (A – B – C – D – A) as having a ping pong pair (A-D).
Also (A – B – A) having a ping-pong pair (A,B).

We got as a result that we have 1201 cases out of 7456 cases (16%) of the cases performing ping-pong behavior.

```
Number of traces with ping-pong: 1201
     case_seq_num  has_ping_pong                                  ping_pong_pairs
1     1-642714990          True  {(V32 2nd, V37 2nd), (D6, V37 2nd), (D6, V32 2...
2     1-642761396          True                {(V32 2nd, V37 2nd), (D2, V37 2nd)}
5     1-643546500          True                {(V32 2nd, V37 2nd), (D5, V37 2nd)}
7     1-643733486          True                {(V32 2nd, V37 2nd), (D6, V37 2nd)}
8     1-644761840          True  {(V32 2nd, V37 2nd), (V32 2nd, V35 2nd), (V35 ...
...           ...           ...                                                ...
7324  1-740801127          True                                      {(A14, D7)}
7344  1-740811922          True                           {(G236 2nd, N15 2nd)}
7363  1-740816836          True                         {(G97, S11), (G179, G97)}
7426  1-740853009          True                              {(G236 2nd, G92)}
7443  1-740859781          True                               {(G97, S9 2nd)}

[1201 rows x 3 columns]
```

Result from the terminal showing the new dataframe created to check the case_seq_num and ping-pong pairs

Now, from the new DataFrame, we can inspect all pairs performing ping-pong, and count them to identify the most common pairs of teams involved in the ping-pong behavior.

```
all_pairs = [pair for pairs in ping_pong_df["ping_pong_pairs"] for pair in pairs]
pair_counts  = Counter(all_pairs)
tt = sum(pair_counts.values())
print("Most common ping pong pairs")
for (a,b) , count in pair_counts.most_common(5):
    print(f"{a} <-> {b}: {count} times, {(count / tt) * 100 :.2f}%")

all_orgs = []
all_orgs = [org    for pairs in ping_pong_df["ping_pong_pairs"] for pair in pairs for org in pair]
orgs_counts  = Counter(all_orgs)
tt = sum(orgs_counts.values())
print("Most common ping pong org:groups")
for org , count in orgs_counts.most_common(5):
    print(f"{org}: {count} traces, {(count / tt) * 100 :.2f}%")
```

Code to print the most ping-pong pairs with the percentage

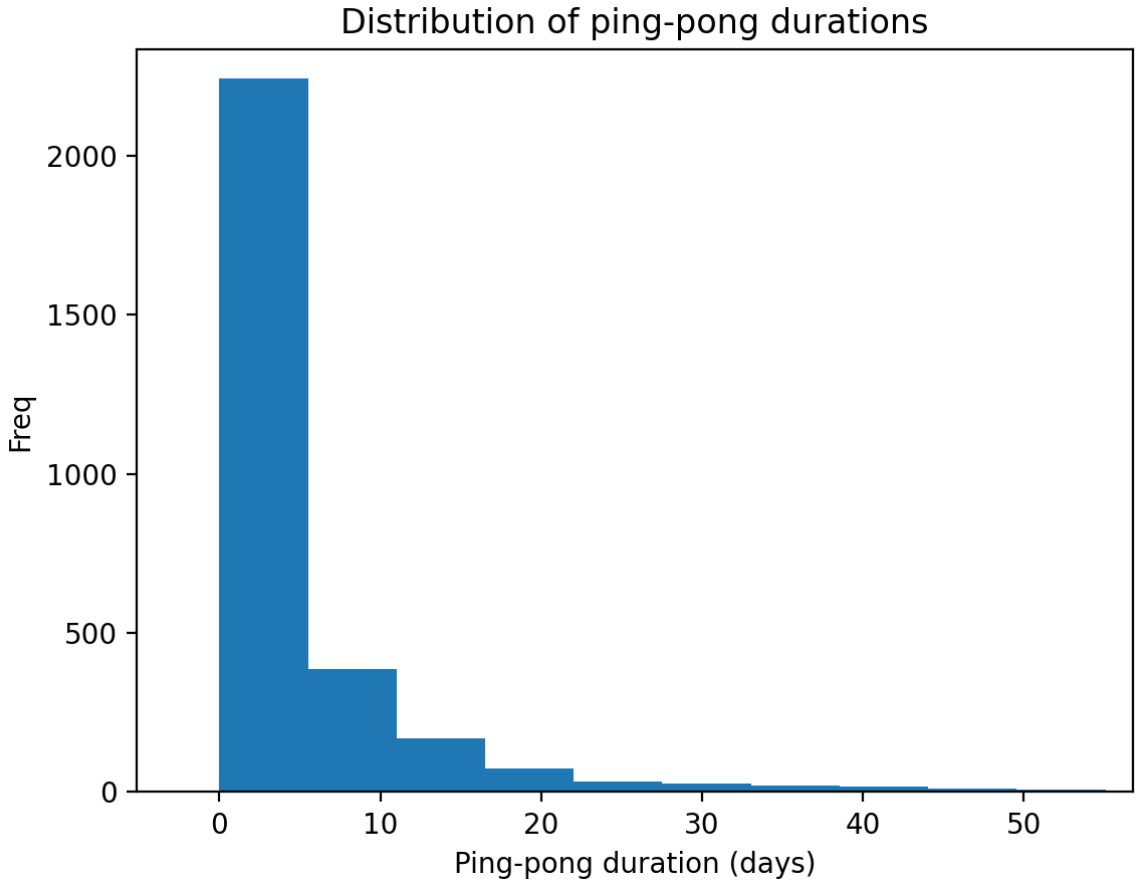
```
Most common ping pong pairs
D4 <-> D5: 57/1819 times, 3.13%
D4 <-> N26 2nd: 50/1819 times, 2.75%
D8 <-> G179: 40/1819 times, 2.20%
G230 2nd <-> G97: 36/1819 times, 1.98%
D4 <-> D7: 35/1819 times, 1.92%
389
Most common ping pong org:groups
D4: 315/3638 traces, 8.66%
G97: 285/3638 traces, 7.83%
D5: 145/3638 traces, 3.99%
D2: 116/3638 traces, 3.19%
D8: 114/3638 traces, 3.13%
```

Terminal output showing the results of most common pairs with their percentage

We can see from the results that the most common pair is D4-D5 as it appears 57 times, and D4-N26 as it appears 50 times.
We observed also that D4 (8.66%) and G97 (7.83%) are the most common teams that are performing ping-pong behavior.

2-2- Quantify the frequency and duration of ping-pong behavior:

Other observations that we can inspect from our data is to inspect the durations between the ping-pong pairs. The duration is calculated in days between the timestamp of the event performing the ping until the group is again taking the case.
We made a histogram chart using matplotlib library to show the distribution of the durations of the ping-pong.



Histogram plot shows the durations of the ping-pong (zoomed-in)

We notice that we have 3026 pairs of ping-pong cases, and we can see that the average duration of the ping-pong cases is 5 days which is for 75% of the cases.
Maximum duration is 110 days which we should observe those pairs.

	duration
count	3026.000000
mean	5.376455
std	11.406737
min	0.000243
25%	0.156881
50%	1.188814
75%	5.833333
max	110.123669

We checked the top 5 pairs having longest durations and we found the following:

```
Top 5 ping pong pairs by max duration (with case):
Case 1-647788578: D5 <-> V37 2nd : 110.12 days
Case 1-642714990: V37 2nd <-> V32 2nd : 109.03 days
Case 1-651196130: D6 <-> V46 2nd : 105.88 days
Case 1-655281402: G18 3rd <-> L23 3rd : 105.10 days
Case 1-652256138: D2 <-> V51 2nd : 103.88 days
```

Terminal output showing top 5 pairs having longest durations

D5-V372nd pair is doing ping pong case which takes 110 days

	org:grou	resource	organiza	org:reso	organiza	org:role	concept	impact	product	lifecycle	time:timestamp	case:co	pt:name
1670	D5	Belgium	be	Katia	Org line C	A2_1	Accepted	Medium	PROD542	In Progress	2012-01-19 10:36:47+00:00	1-647788578	
1671	D5	Belgium	be	Katia	Org line C	A2_1	Accepted	Medium	PROD542	In Progress	2012-01-19 10:37:28+00:00	1-647788578	
1672	D5	Belgium	be	Katia	Org line C	A2_1	Accepted	Medium	PROD542	In Progress	2012-01-19 10:45:57+00:00	1-647788578	
1673	D4	Belgium	be	Katia	Org line A2	A2_1	Queued	Medium	PROD542	Awaiting As	2012-01-19 10:46:08+00:00	1-647788578	
1674	D5	Belgium	be	Katia	Org line C	A2_1	Accepted	Medium	PROD542	In Progress	2012-01-19 10:46:38+00:00	1-647788578	
1675	V37 2nd	Belgium	be	Katia	Org line V7n		Queued	Medium	PROD542	Awaiting As	2012-01-19 10:47:36+00:00	1-647788578	
1676	V37 2nd	Netherland	be	Danny	Org line V7n		Accepted	Medium	PROD542	In Progress	2012-01-19 15:43:58+00:00	1-647788578	
1677	V32 2nd	Netherland	be	Danny	Org line V7n		Queued	Medium	PROD542	Awaiting As	2012-01-19 16:41:15+00:00	1-647788578	
1678	V32 2nd	Netherland	be	Earl	Org line V7n		Accepted	Medium	PROD542	In Progress	2012-03-27 15:58:10+00:00	1-647788578	
1679	V37 2nd	Netherland	be	Juan	Org line V7n		Accepted	Medium	PROD542	In Progress	2012-05-04 15:06:27+00:00	1-647788578	
1680	D5	Netherland	be	Danny	Org line C	A2_1	Queued	Medium	PROD542	Awaiting As	2012-05-08 13:44:43+00:00	1-647788578	
1681	D5	Belgium	be	Katia	Org line C	A2_1	Accepted	Medium	PROD542	In Progress	2012-05-08 15:49:09+00:00	1-647788578	
1682	D5	Belgium	be	Katia	Org line C	A2_1	Accepted	Medium	PROD542	Wait - User	2012-05-08 15:49:21+00:00	1-647788578	
1683	D5	Belgium	be	Katia	Org line C	A2_1	Accepted	Medium	PROD542	Wait - User	2012-05-08 15:50:17+00:00	1-647788578	
1684	D5	Belgium	be	Katia	Org line C	A2_1	Completed	Medium	PROD542	Resolved	2012-05-09 08:29:07+00:00	1-647788578	
1685	D5	Belgium	be	Katia	Org line C	A2_1	Completed	Medium	PROD542	Closed	2012-05-09 08:29:09+00:00	1-647788578	

Excel spreadsheet showing stack trace of the case that taking the maximum days (110.2 days)

```
82
83  all_durations = [pair[1] for pairs in ping_pong_df["ping_pong_pairs"] for pair in pairs]
84
85  plt.hist(all_durations, bins=20)
86  plt.xlabel("Ping-pong duration (days)")
87  plt.ylabel("Freq")
88  plt.title("Distribution of ping-pong durations")
89  plt.show()
90
91
92  dd = pd.DataFrame(all_durations, columns=["duration"])
93  dd.describe()
94
95  pair_df = pd.DataFrame(
96      [
97          (row.case_seq_num, p, d)
98          for _, row in ping_pong_df.iterrows()
99          for (p, d) in row.ping_pong_pairs
100     ],
101     columns=["case_seq_num", "pair", "duration"]
102 )
103 idx_max = pair_df.groupby("pair")["duration"].idxmax()
104 max_cases = (
105     pair_df
106     .loc[idx_max, ["case_seq_num", "pair", "duration"]]
107     .sort_values("duration", ascending=False)
108     .head(10)                       # top 10 pairs, change to .head(5) for top 5
109     .reset_index(drop=True)
110 )
111
112 print("Top 5 ping pong pairs by max duration (with case):")
113 for _, row in max_cases.head(5).iterrows():
114     a, b = row["pair"]
115     print(f"Case {row['case_seq_num']}: {a} <-> {b} : {row['duration']:.2f} days")
116
```

3- Cross department conformance across different departments:

In step 1.2-Log segmentations, we discovered that organizations "ORG-C" and "ORG-A2" are handling most of the cases in our event log (65%, 18%).
We decided to do conformance checking on the events of those 2 logs.

3-1- Clean the logs:

After we have the logs of the organizations ORG-C and ORG-A2, we noticed that we have some traces which are not completed. In order to focus the discovery on the core incident-management steps, and to model the full incident management lifecycle, we did the filter steps to the logs of each department:

A. Drop cases where the last concept:name is not Completed. For org-C we dropped 969 cases out of 6080 cases and for org-A2 we dropped 453 out of 1740 cases.
B. In each trace, we are dropping the consecutive duplicates events.
C. Rename the columns names.

```
cases_before = df["case_seq_num"].nunique()
df = df.sort_values(["case_seq_num", "timestamp"])
df = df.groupby("case_seq_num").filter(lambda trace: trace["concept_name"].iloc[-1] == "Completed").reset_index(drop=True)
cases_after = df["case_seq_num"].nunique()
dropped = cases_before - cases_after
print(f"Dropped {dropped} cases out of {cases_before}  for not having Completed as last concept_name ")

df = df[df["concept_name"].ne(df.groupby("case_seq_num")["concept_name"].shift())].reset_index(drop=True)
df = df[["case_seq_num", "concept_name", "timestamp"]]

df = df.rename(columns={
    "case_seq_num": "case:concept:name",
    "concept_name": "concept:name",
    "timestamp": "time:timestamp"
})
df = df.sort_values(["case:concept:name", "concept:name", "time:timestamp"])
df = dataframe_utils.convert_timestamp_columns_in_df(df)
```

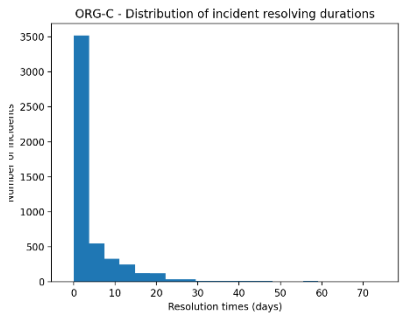
Code snippet to drop cases that are not completed, drop consecutive dupicates, and rename columns.

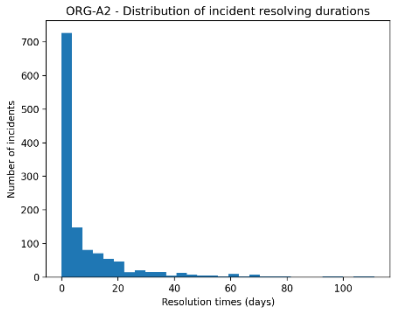
3-2- Statistics about resolving durations:

We calculate the durations in days for each department to close/resolve a ticket. Afterward, we are showing a histogram plots illustrating those durations, and a table showing summary of durations (min,max, mean, 25%, 50%, 75%).

```
67  def get_resolution_durations(df,fig_title):
68      """
69          Return
70              Statistics of duration to resolve a ticket for the dataframe
71              Histogram Plot showing duration in days for resolving
72      """
73      df = df.sort_values(["case:concept:name", "time:timestamp"])
74      df["time:timestamp"] = pd.to_datetime(df["time:timestamp"])
75      dur = df.groupby("case:concept:name")["time:timestamp"].agg(start="min", end="max").reset_index()
76      dur["duration"] = dur["end"] - dur["start"]
77      dur["duration_days"] = dur["duration"].dt.total_seconds() /  (24*60*60)
78      stats = dur["duration_days"].describe()
79      stats_df = stats.to_frame().T
80
81      plt.figure()
82      plt.hist(dur['duration_days'], bins=30)
83      plt.xlabel("Resolution times (days)")
84      plt.ylabel("Number of incidents")
85      plt.title(fig_title)
86
87      return stats_df, plt
88
```

Calculating durations of each case trace and showing the histogram





The table below shows the durations statistics in days of Org-a2 and org-c

(days)	Org-A2	Org-C
Count	1287	5111
Mean	9.68	4.69
Min	0.0	10.35
25%	0.12	0.006
50%	2.05	0.06
75%	11.6	5.75
Max	111.07	110.91

We noticed that org-C is resolving the tickets in much faster way than org-a2

3-3- Discover the most common traces for each department:

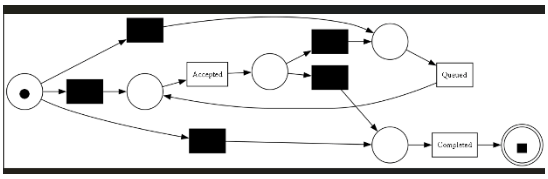
For each department we discover the most common traces for each department, the result as in the picture:

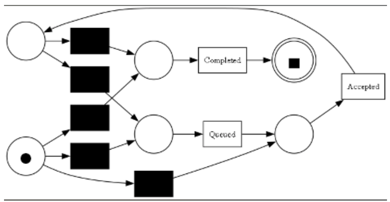
```
Most common traces in ORG-A2 after filtering
concept:name
(Queued, Accepted, Completed)                                                                                                         0.409479
(Accepted, Completed)                                                                                                                 0.215229
(Accepted, Queued, Accepted, Completed)                                                                                               0.114219
(Queued, Accepted, Queued, Accepted, Completed)                                                                                       0.081585
(Accepted, Queued, Accepted, Queued, Accepted, Completed)                                                                             0.052836
(Queued, Accepted, Queued, Accepted, Queued, Accepted, Completed)                                                                     0.017871
(Accepted, Queued, Accepted, Queued, Accepted, Queued, Accepted, Queued, Accepted, Completed)                                         0.017094
(Accepted, Queued, Accepted, Queued, Accepted, Queued, Accepted, Completed)                                                           0.015540
(Queued, Accepted, Completed, Queued, Accepted, Completed)                                                                            0.006216
(Accepted, Queued, Accepted, Queued, Accepted, Queued, Accepted, Queued, Accepted, Queued, Accepted, Queued, Accepted, Queued, Accepted, Queued, Accepted, Completed)    0.005439
Name: proportion, dtype: float64
Most common traces in ORG-C after filtering
concept:name
(Accepted, Completed)                                                                             0.478380
(Accepted, Queued, Accepted, Completed)                                                           0.201526
(Queued, Accepted, Completed)                                                                     0.081002
(Accepted, Queued, Accepted, Queued, Accepted, Completed)                                         0.073567
(Accepted, Queued, Accepted, Queued, Accepted, Queued, Accepted, Completed)                       0.029740
(Queued, Accepted, Queued, Accepted, Completed)                                                   0.019761
(Accepted, Completed, Accepted, Completed)                                                        0.018979
(Accepted, Queued, Accepted, Queued, Accepted, Queued, Accepted, Queued, Accepted, Completed)     0.013305
(Queued, Accepted, Queued, Accepted, Queued, Accepted, Completed)                                 0.008609
(Accepted, Queued, Accepted, Queued, Accepted, Queued, Accepted, Queued, Accepted, Queued, Accepted, Completed)    0.007239
Name: proportion, dtype: float64
```

We notice that 40% of the cases in org-a2 follows the path (Queued - Accepted - Completed) and 47% of the cases in org-C are following (Accepted - Completed).

3-4- Petri-net:

For each department we extract the petri-net model by using Heuristic Miner which captures the department's core workflow, then we visualize the petri-nets.






```python
def get_petri(df):
    df = pm4py.format_dataframe(
        df,
        case_id="case:concept:name",
        activity_key="concept:name",
        timestamp_key="time:timestamp",
        timest_format=None
    )
    lg = log_converter.apply(df, variant=log_converter.Variants.TO_EVENT_LOG)
    net, im, fm = heuristc_miner(lg)
    return net, im, fm, lg

def visualize_petri(net, im, fm):
    gviz = vis_factory(net, im, fm)
    pn_visualizer(gviz)
```

3-5- Compute alignment-based fitness:

We compute alignment-based fitness scores by replaying each department's event log on the other department's petri-net model, and after calculating a quantitive measure (0-1) of how well their process executions conform to each other.

```
116    def get_alignments_fitness(log_1, net_2, im_2, fm_2):
117        fit = alignments_apply(log_1, net_2, im_2, fm_2)[0]["fitness"]
118        return fit
```

Function to calculate the fitness, takes as parameter the logs of org_1 and petri-net info of org_2

We found out that the fitness value scores for applying the model of A2 on C and also applying the C model on A2 is 1.00.

```
Fitness A2 -> C: 1.0
Fitness C -> A2: 1.0
```

Terminal output for the fitness values

3-6- Try to build the model based on "lifecycle:transition" instead of "concept:name":

We wanted also to try discover the model process but depending on the *lifecycle_transition* instead of *concept_name.*

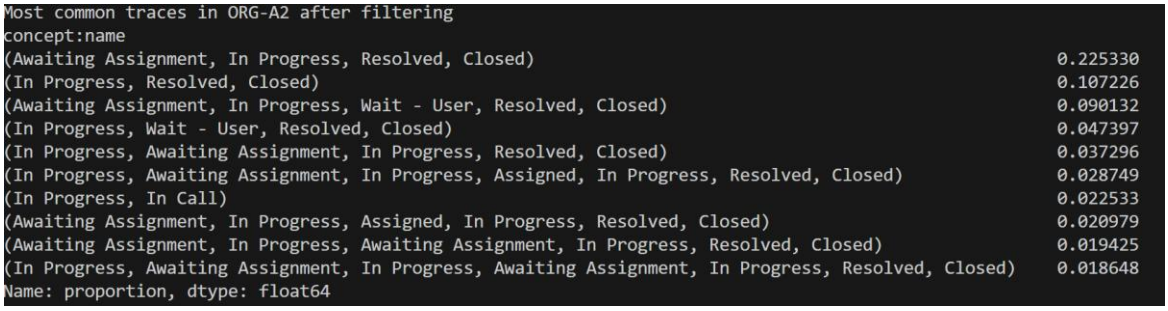
Concept_name values are: ['Queued', 'Accepted', 'Completed']

Lifecycle_transition values are: ['Awaiting Assignment', 'In Progress', 'Wait - User', 'Resolved', 'Closed', 'Wait', 'Assigned', 'In Call'].

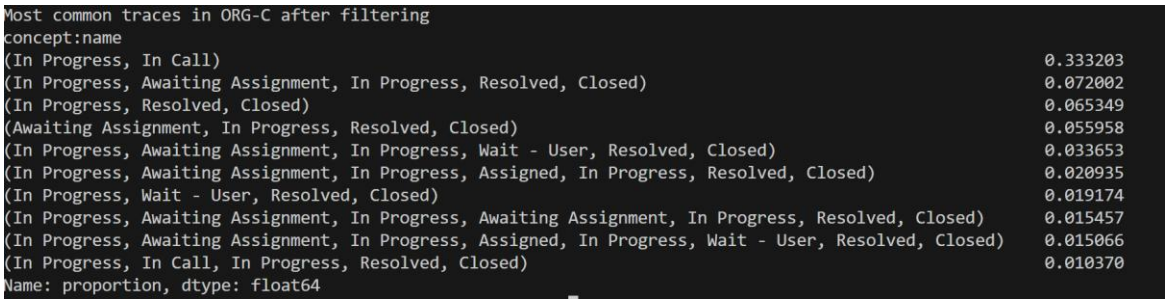
Before in step 3.1.A, we dropped cases where their last concept_name is not Completed, now we want to drop cases where their last lifecycle_transition is not one of ["Resolved", "Closed", "Cancelled", "In Call"]

```
Dropped 453 cases out of 1740  for not having ['Resolved', 'Closed', 'Cancelled', 'In Call'] as last lifecycle_transition in their trace
Dropped 969 cases out of 6080  for not having ['Resolved', 'Closed', 'Cancelled', 'In Call'] as last lifecycle_transition in their trace
```

Terminal output of dropping not-ending cases

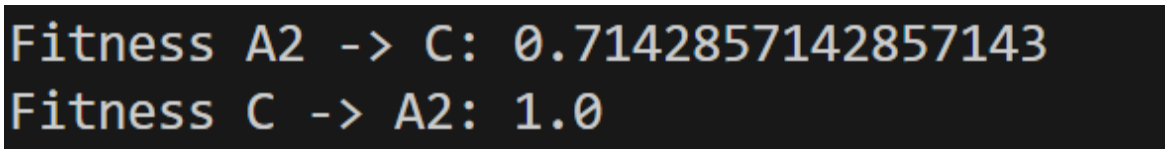


Terminal output showing most common traces in Org-A2



Terminal output showing most common traces in Org-C

After we calculate the fitting results of applying one model of a department on the other's department log:



We can notice that only about 71% of org-a2 events can be perfectly replayed by org-c's petri-net model.

By contrast, a fitness of 1.9 for c-> a2 whichh means that all of org-c's events fit org-a2's model exactly.

Org-a2 sometimes follow paths that org-c never does in its "core" process. Org-c never does anything outside of what A2 does.

4- Identify potential improvements:

4-1- Better ticket routing (reduce ping-pong):

A major cause of ping-pong behavior is wrong assignments of tickets. We recommend implementing an improved routing mechanism so that each incident is assigned to the correct group on the first try. This could be by using ML to

automate the assigning process based on historical data which can lead to limit the number of ping-pong cases.

4-2- Improve handover protocols:

Every handoff should add a value and not just shuffle responsibility. we suggest creating a special handover protocol: when escalating the ticket, the current team must provides sufficient details and specific task for the receiving team. Likewise, when the team sends the ticket back, they should clearly state the reason to avoid pong behaviors.

4-3- Share knowledge:

Obviously from the resolving durations statistics, we can observe that the team of org-c is more capable of resolving issues than org-a2, sharing the knowledge with better documentation could improve the traces of org-a2.