VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
HO CHI MINH UNIVERSITY OF TECHNOLOGY
FACULTY OF APPLIED SCIENCE

MATHEMATICAL MODELING (CO2011)

PETRI-NET MODELING

Instructor: Nguyễn An Khương
Performer: Nguyễn Đắc Hoàng Phú - 2010514
Đoàn Trần Cao Trí - 2010733
Du Thành Đạt - 2010206
Lâm Nhật Tân - 2010597
Lý Gia Huy - 2010289

Ho Chi Minh City,November /2021

1 EXECUTIVE SUMMARY

1.1 Introduction

Through the assignment, we have earned lots of knowledge about information systems. How exactly it helps in controlling and supporting business processes as a discrete dynamic system, which can be modeled as a transition system. As a modeling tool, however, transition systems are not suitable, because it's hard to describe some complex system in terms of state place and transition relation. So, to improve those disadvantages we chose Petri nets to solve these problems.

Petri net is a powerful modeling formalism in computer science, system engineering, and many other disciplines. Petri net combines a well-defined mathematical theory with a graphical representation of the dynamic behavior of systems, the theoretic aspect of Petri net allows precise modeling and analysis of system behavior, while the graphical representation of Petri net enable visualization of the modeled system state changes.

1.2 Problem

Our mission: Under a SARS pandemic where a huge lack of ICU beds occurs in city H, patients should consult specialists in the outpatient clinic of a hospital, we describe the course of business around a specialist in this outpatient clinic of hospital X as a process model, formally, we use Petri Net.

1.3 Objective

We model this business process (step by step) as a Petri net by using C++ code and visualizing it on website.

1.4 Team

No	Fullname	Student ID	Role	Percentage of work
1	Nguyễn Đắc Hoàng Phú	2010514	Leader	100%
2	Du Thành Đạt	2010206	Member	100%
3	Đoàn Trần Cao Trí	2010733	Member	100%
4	Lý Gia Huy	2010289	Member	100%
5	Lâm Nhật Tân	2010597	Member	100%

1.5 Files

File Name	Explain
.pdf	Report file, log
.tex	Latex file
.cpp	Coding file (C++)
.html	Web Visualization

2 TABLE OF CONTENTS

Contents

3 SUMMARY THEORY

3.1 Definition

A Petri net is a triple (P,T,F), where

1. P is a finite set of places
2. T is a finite set of transitions.
3. F $\subseteq (P * T) \bigcup (T * P)$ is a flow relation.

Consequently, there is a one-to-one correspondence between the graphical representation of a Petri net and the triple (P, T, F).

A Petri net system (P, T, F, M_0) consists of a Petri net (P, T, F) and a distinguished marking M_0, the inital marking.

3.2 Marking

A marking of a Petri net is a function m : P $\rightarrow \mathbb{N}$, assigning to each place p $\in$ P the number m(p) of tokens at this place. The set M of all markings of this net is the set of all such functions.

We can now define the concept of enabling. A transition t is enabled at marking m if every input place of t contains at least one token.

3.3 Enabling

In a Petri net (P, T, F), a transition t $\in$ T is enabled at marking m: P $\rightarrow \mathbb{N}$ if and only if for all p $\in tokens, m(p) > 0$.

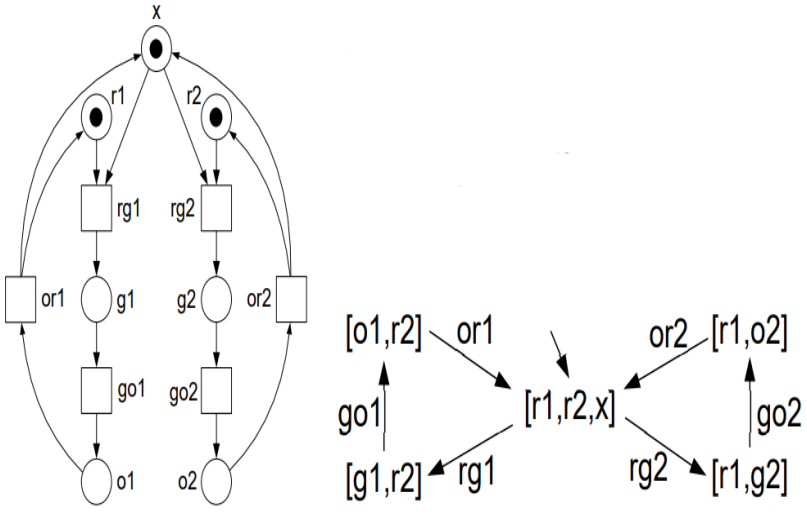
3.4 Transition firing

For a Petri net (P, T, F), let w be the weight function and m: P $\rightarrow \mathbb{N}$ be the current marking. A transition t $\in$ T can fire if and only if it is enabled at m. The firing of t yields a new marking m': P $\rightarrow \mathbb{N}$ where for all place p $\in$ P, $m'(p) = m(p) - w((p,t)) + w((t,p))$.

An enabled transition can fire by consuming a token from each input place and producing a token for each output place.

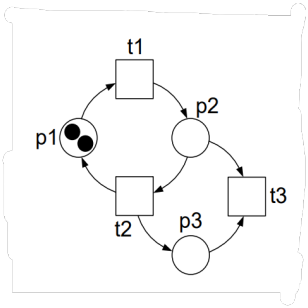
3.5 Reachability graph

The reachability graph of a Petri net is a transition system with one initial state (initial marking) and no explicit final marking.



3.6 Boundedness

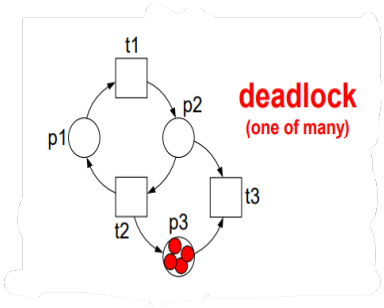
- A place p is k-bounded if there is no reachable marking with more than k tokens in p.
- A Petri net is k-bounded if all of places are k-bounded.
- A place or Petri net is bounded if there exists such a k.



This Petri net is unbounded because $p3$ is unbounded.

3.7 Deadlock

- A marking is dead if there no transition is enabled in it.
- A Petri net has a potential deadlock if there is a reachable dead marking.
- A Petri net is deadlock-free if each reachable marking enables at least one transition.



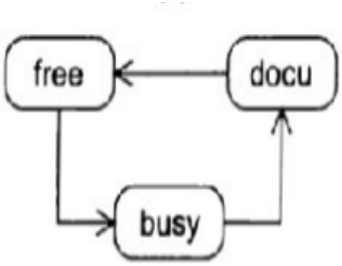
3.8 Modeling

In the outpatient clinic of a hospital, patients consult specialists. Each patient has an appointment with a curtain specialist. We describe the course of business around a specialist as a process model. As a formalism, we can see:

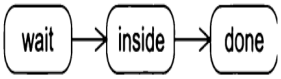
- The specialist receives patient per period. At each moment, the specialist is in one of the following three states:
  1. The specialist is free and waits for the next patient **(state free)**;
  2. The specialist is busy treating a patient **(state busy)**;
  3. The specialist is documenting the result of the treatment **(state docu)**.
- Every patient who visits a specialist is in one of the following three states:
  1. The patient is waiting **(state wait)**;
  2. The patient is treating by the specialist **(state inside)**;
  3. The patient had been already treated by the specialist **(state done)**.

In this model, we define the states by person as:

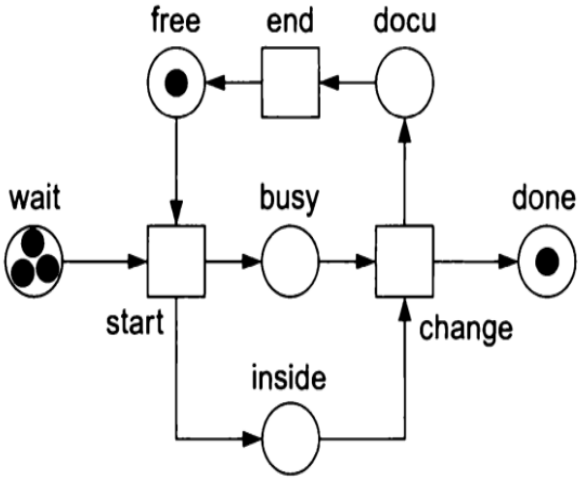
1. **Specialist**: Each patient has an appointment with a certain specialist. The specialist receives patient per period. At each moment, the specialist is in one of the following three states. The specialist goes through the three states in an iterative way.



2. **Patient**: who visits a specialist is in one of the following three states. A patient goes through these states only once (per visit).



By using superimposition operator two Petri nets above, we receive this Petri net to describe the course of business around a specialist in this outpatient clinic of hospital X



- Transition *start*, *change*, and *end* represents the three possible events.
- The tokens in place *wait* represent the number of waiting patients.
- A token in place *inside* represents a patient who is being treated or examined by the specialist.
- The tokens in place done represent the patients who had been treated or examined.
- The token in place *free* indicates that the specialist is in free state.
- If a specialist is busy treating a patient, a token will be in place busy.
- A token in place *docu* indicates that the specialist is documenting certain results.

4 CODE EXPLANATION

4.1 LIBRARY

```
#include <iostream>
#include <vector>
#include <windows.h>
```

Note:

+ The library <**iostream**> is standard input / output streams library.
+ We store the state transitions by <**vector**>.
+ The library <**windows.h**> is a Windows-specific header file for the C and C++ programming languages which contains declarations for all of the functions in the Windows API.

4.2 SOURCE CODE

4.2.1 CLASS

```
//-------global variable & class prototype----------
class Transition;
class Place;
vector<Place*> setOfPlace;
vector<Transition*> setOfTransition;
bool tokenRun = true;
//----------------------------------------------------

//create a frame so that the output is contained in that frame when printing to the console
void gotoXY(int column, int line) {
   COORD coord;
   coord.X = column;
   coord.Y = line;
   SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}
```

The class **Transition** and **Place** will be described after. Two vectors **setOfPlace** and **setOfTransition** are used to store the state. Boolean variable **tokenRun** is created to check if there is any change in the progress. Function **gotoXY** makes the printed model go back to **(0; 0)** position and continue printing.

```
class Place
{
//method
public:
   Place() {
      this->token = 0;
      setOfPlace.push_back(this);
   };
   ~Place() {
      this->token = 0;
      this->inDegree.clear();
      this->outDegree.clear();
   };
   void addToken() {
      this->token++;
   }
   void addToken(int t) {
      this->token += t;
   }
```

```
   void flowTransition();
   void setFlowTo(Transition* tr);
//member
public:
   unsigned int token;
   vector<Transition*> inDegree;
   vector<Transition*> outDegree;
};
```

The constructor **Place()** sets the token to zero and pushes back this **Place** to **setOfPlace**. The destructor **Place()** also sets the token to zero and clear vector **inDegree** and **outDegree**. **inDegree** is the number of transition connect to this place, and the **outDegree** is opposite. The functions **flowTransition** and **setFlowto** are described below.

```
1  void Place::flowTransition() {
2     int fig = 0;
3     for (auto i : this->outDegree) {
4        if (i->flag) {
5           fig++;
6        }
7     }
8     if (token >= fig) {
9        for (auto i : this->outDegree) {
10          if (i->flag) {
11             token--;
12             i->token++;
13             tokenRun = true;
14          }
15       }
16    }
17 }
```

The first for-loop in line 3 counts the number of transition accept to flow, Then the program checks if this place has enough token to it's out transition by the if-function in line 8.

```
1  void Place::setFlowTo(Transition* tr) {
2     this->outDegree.push_back(tr);
3     tr->inDegree.push_back(this);
4  }
```

```
1  class Transition
2  {
3  //method
4  public:
5     Transition() {
6        this->flag = false;
7        this->token = 0;
8        setOfTransition.push_back(this);
9     };
10    ~Transition() {
11       this->token = 0;
12       inDegree.clear();
13       outDegree.clear();
14    };
15    void firing();
16    void setFlowTo(Place* p);
17    friend void acceptFlow();
18 //member
19 public:
20    unsigned int token;
```

```
   bool flag;
   vector<Place*> inDegree;
   vector<Place*> outDegree;
};
```

The constructor **Transition()** sets the **flag** to false, the **token** to zero and pushes back this **Transition** to **setOfTransition**. The destructor also reset the token to zero and clear the **inDegree** and **outDegree**. The function **firing**, **setFlowto** and **accceptFlow** are described below.

```
void Transition::firing() {
   if (this->token == inDegree.size()) {
      this->token = 0;
      for (auto i : outDegree) i->token++;
   }
}
```

First, we check if the amount of token of this Place is equal to the size of **inDegree**. Then, flow one token to each out place.

```
void Transition::setFlowTo(Place* p) {
   this->outDegree.push_back(p);
   p->inDegree.push_back(this);
}
```

This function is the same as the above.

```
void acceptFlow() {
   for (auto j : setOfTransition) {
      bool check = true;
      for (auto i : j->inDegree) {
         if (i->token == 0) {
            check = false;
            break;
         }
      }
      j->flag = check;
   }
}
```

First, the program loops end checks all transitions. If nothing happens, this transition is accepted to flow. For each transition, we must go to check it's **inDegree** place, if there is at least one place missing the token, the for-loop will be broken.

```
void PRINT(int free, int wait, int busy, int inside, int done, int docu) {
   gotoXY(0, 0);
   printf("      ( %d)<---[]<---( %d)     \n", free, docu);
   printf("       |             ^         \n");
   printf("       v             |         \n");
   printf("( %d)--->[]--->( %d)--->[]--->( %d)\n", wait, busy, done);
   printf("       |             ^         \n");
   printf("       v             |         \n");
   printf("        \\--->( %d)----/      \n",inside);
}
```

The format printed result of the model is above.

4.2.2 MAIN

```
//----------------------------------------
   //initialize place "free" and "wait"
```

```
   Place* free = new Place();
   Place* wait = new Place();

    //initialize transition "start"
   Transition* start = new Transition();

   free->setFlowTo(start);
   wait->setFlowTo(start);

   //initialize place "busy" and "inside"
   Place* busy = new Place();
   Place* inside = new Place();

   start->setFlowTo(busy);
   start->setFlowTo(inside);
//----------------------------------------

//----------------------------------------
   //initialize transition "change"
   Transition* change = new Transition();

   busy->setFlowTo(change);
   inside->setFlowTo(change);

   //initialize "done" and "docu" place
   Place* done = new Place();
   Place* docu = new Place();

   change->setFlowTo(done);
   change->setFlowTo(docu);
//----------------------------------------

//----------------------------------------
    #initialize "end" transition
   Transition* end = new Transition();

   docu->setFlowTo(end);
   end->setFlowTo(free);
//----------------------------------------
```

First, we must create and set the flow as above.

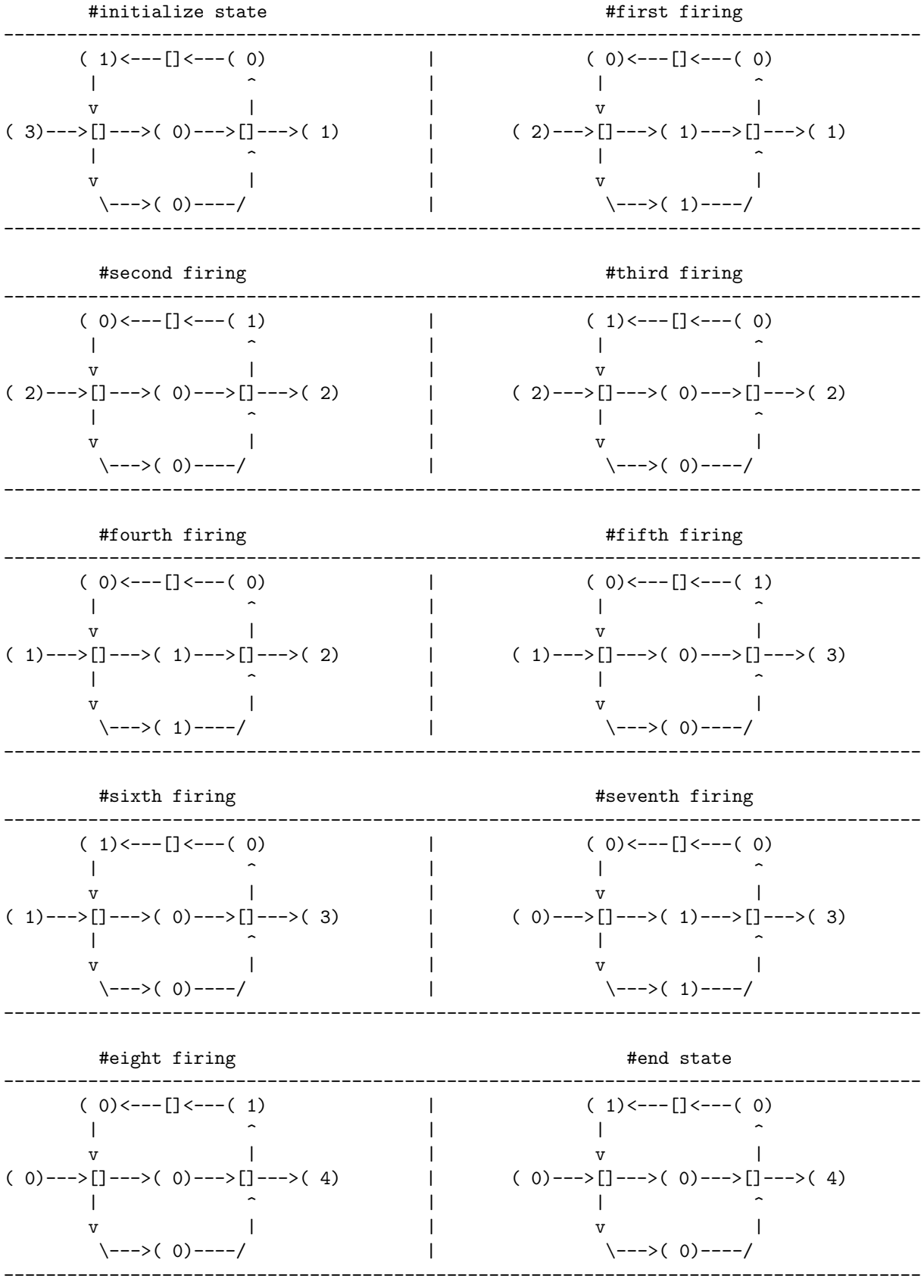
```
   free->addToken(1);
   wait->addToken(3);
   done->addToken(1);
```

After that, we add the token from input given.

```
   do {
      tokenRun = false;
      PRINT(free->token, wait->token, busy->token, inside->token, done->token, docu->token);
      acceptFlow();
      for (auto i : setOfPlace) i->flowTransition();
      Sleep(1000);
      for (auto i : setOfTransition) i->firing();
      Sleep(2000);
   } while (tokenRun);
```

We must set the default **false** this process to **tokenRun**, there are nothing change. Then, we set up the flag of all transitions and go through each place and flow or not, similar to fire each transition. If there exist any change, this model keeps going into the loop.

4.3 DEMO


#initialize state
#first firing
#second firing
#third firing
#fourth firing
#fifth firing
#sixth firing
#seventh firing
#eight firing
#end state

5 PROBLEM SOLUTION

5.1 Question 1.

Given the Petri net N_S modeling the state of the specialist:

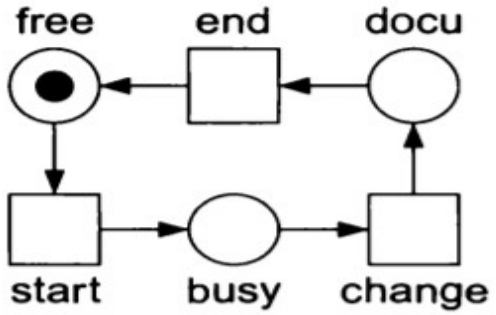


Figure 1: *A Petri net modeling state of specialists.*

1. **Question 1a.** In the Petri net N_S:
   + Place: free, busy, docu.
   + Transition: start, change, end.
2. **Question 1b.** The system could be represented by the marking of the net as a triple (x, y, z) with x specifying the number of tokens in state *free*, y in state *busy*, and z in state *docu*:

$$(1, 0, 0) \longrightarrow (0, 1, 0) \longrightarrow (0, 0, 1) \longrightarrow (1, 0, 0) \longrightarrow ...$$

5.2 Question 2.

Define N_{Pa} as the Petri net modeling the state of patients:

1. **Question a.** The possible meaning of a token in state *inside* of the net: The state *inside* contains one patient treated by the specialist.
2. **Question b.** Construct the Petri net N_{Pa} assumes that there are five patients in state *wait*, no patient in state *inside*, and one patient is in state *done*:

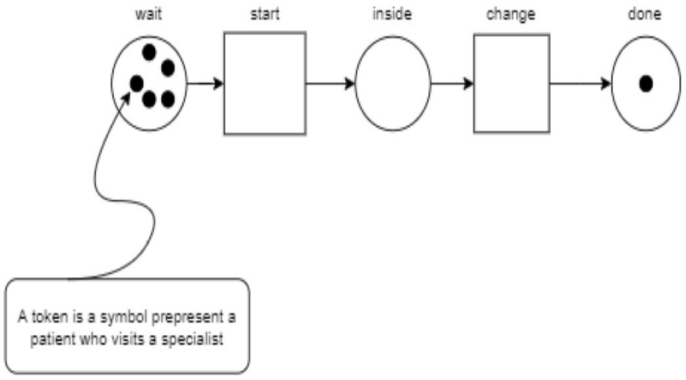


Figure 2: *A Petri net modeling state of patients.*

5.3 Question 3.

Determine the superimposed (merged) Petri net N_S and N_{Pa} allows specialist to treat patients, assume that there are four patients who are waiting for specialist/doctor, a patient in state *done*, and a specialist in state *free*:

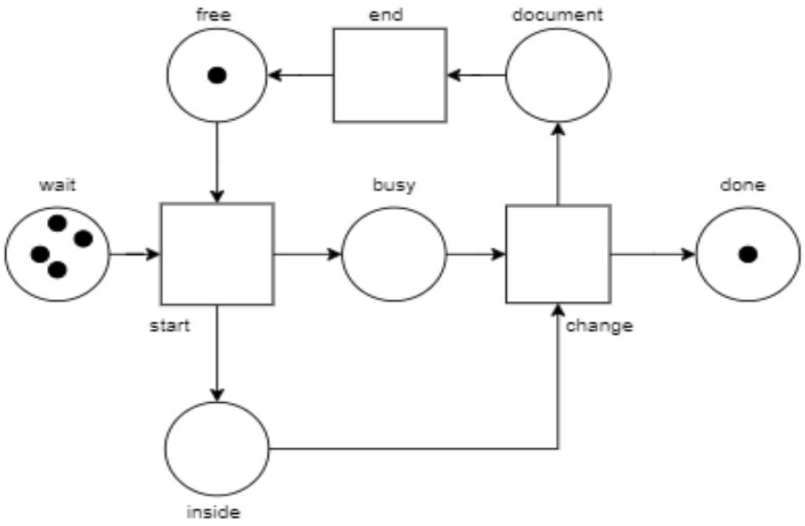


Figure 3: *A merged Petri net modeling state of patients and specialists.*

5.4 Question 4.

Consider an initial marking $M_0 = [3.wait, done, free]$ in the merged Petri net N. The marking to be reachable from M_0 by firing one transition once is $M = [2.wait, inside, done, busy]$. Because only a transition *start* was fired once, so one token in *wait* and one token in *free* are changed, they are go to *inside* and *busy*.

5.5 Question 5.

The superimposed Petri net N is a **deadlock free** Petri net if the number of tokens in *busy* is different from another in *inside*. Explain:

- Each time a token in *free* is done a process, the number of tokens in *wait* reduce 1 and in *done* increase 1.
- The number of tokens in *wait* is finite, until in *wait* will not contain any token, so firing transition *start* is an impossible problem, causing some tokens in *free* is not be free after firing. Leading to transition as *start*, *change*, *end* could not be fired.

5.6 Question 6.

Propose a similar Petri net with two specialists already for treating patients, with explicitly explained construction.

In reality, with more than one specialist, the process can take place in parallel. But in this situation, we propose that there has only one treating room and specialist in this room is*busy* treating while a patient is being treated (*inside stage*). So a process will take place in sequence.

According to initial marking, we have two specialists in *free* place. At each time, a place flows one token to every transition which connected to it. Without loss of generality, assume that specialist with smallest IDs go treating first. In Petri Net, the above assumption is an application of Coloured Petri

Net, that mean we can determine specialists by their IDs, the simplest way is to add data (IDs in this situation).

Similar with patients, a patient who arrive first will get smaller IDs value than (from zero). Because we know a patient who arrive soon due to an attribute of time clock. In these arguments, we also assume that the order of service for patients with smaller IDs takes precedence.

In each such area, there will be 3 cases corresponding to the paths defined above:

1. **Case 1:** Where there is no token, this area is not taking any action
2. **Case 2:** In which there is more than 2 tokens. At this point, there will be 2 more sub-cases: those tokens represent which patients in this place? Or similar question to specialists. For convenience, we add IDs as follow: if the token represents the patient - denoted Px or the token represents the doctor - denoted Sy (x, y is non-negative number).
3. **Case 3:** We do not include at the ticket counter because there can be as many patients as possible. Including 2 tokens. The case of having 2 tokens of the same doctor only occurs with the initial initialization marking in the free area because subsequent markings are sequential (since each subsequent zone can only contain a maximum of 1 doctor and 1 patient).

Consider an initial marking $M_0 = [3.wait, done, 2.free]$. A process will work as follow:

Marking	Place					
	free*	**busy**	**docu**	**wait**	**inside**	**done***
$M_0 = [3.wait, done, 2.free]$	{S0, S1}	{}	{}	{P1,P2,P3}	{}	{P0}
$M_1 = [2.wait, done, free, busy, inside]$	{S1}	{S0}	{}	{P2,P3}	{P1}	{P0}
$M_2 = [wait, 2.done, busy, inside, document]$	{}	{S1}	{S0}	{P3}	{P2}	{P0, P1}
$M_3 = [wait, 3.done, document, free]$	{S0}	{}	{S1}	{P3}	{}	{P0,P1,P2}
$M_4 = [3.done, free, busy, inside]$	{S1}	{S0}	{}	{}	{P3}	{P0,P1,P2}
$M_5 = [4.done, free, document]$	{S1}	{}	{S0}	{}	{}	{P0,P1,P2,P3}
$M_6 = [4.done, 2.free]$	{S0,S1}	{}	{}	{}	{}	{P0,P1,P2,P3}

(*) Definition of table

+ Example in free*, character '*' mean this stage is a final stage.
+ Each cell is a set of tokens that its place.
+ The columns highlighted in blue represents states of the doctors/specialists.
+ The columns highlighted in orange represents states of the patients.

(*) Explain of table

+ The element has the smallest index in a set which has priority to leave the set first.
+ Transferring the state: the element of the set move next to the column which has same color in the direction from left to right in cycle rule, which mean the element in last column move to first column.

6 PROCESS MINING IN REAL LIFE

6.1 EXAMPLE SCENARIO

For example, someone wants to buy a new laptop. This is a requester.

+ First of all, he has to go to his manager to ask for money to buy a new computer.
+ Afterwards, a request goes to purchasing department, which will looks for the best options.
+ Finally, the computer will be ordered, and supplied by the supplier. Eventually, there's an invoice that will be sent and paid through the financial department.

Figure 4: *Purchase Scenario*

Problem: Now imaging that you are the manager of this purchasing process and you have the following problems.

+ Looking for ways to make the process more **efficient**.
+ Demonstrate that the process **complies** with the purchasing guidelines.
+ Received **complaints** lately about the process duration

Target: So, based on these problems, you derived the following analysis goals

+ **Understanding the process** in detail.
+ **Checking for deviations** from the original principles.
+ **Controling the performance targets** in 21 days.

Why 21 days: It takes 21 days to completely form a new habit, a new activity. Because 21 days is the time it takes for new neural pathways to fully form in your brain.

6.2 ROADMAP

In a mining project, there will be 4 phases:

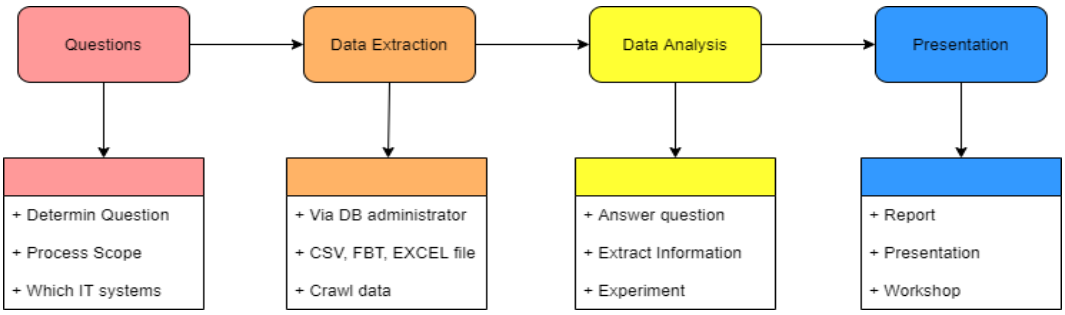


Figure 5: *roadmap*

6.2.1 Phase1: 3 typical questions

+ How does the process actually look like ?
+ Can we achieve the target performance ?
+ Are there any deviations from the prescribed rules?

6.2.2 Phase2: Data Extraction

1. **Data requirements:** You only need to find three distinct pieces
   (a) **Case ID:** Is a process instance identifier.Depending on the type of process there are different types of case IDs. Example: Treatment process – a patient ID; education process – student code,...
   (b) **Activity:** or a status change, or a transaction name. So this is about the steps that are being performed in the process.
   (c) **Timestamp:** bring anything to the right order

Case ID	Activity	Start Time	End Time
339	Create Purchase Requisition	2011/02/16 20:31:00.000	2011/02/16 21:23:00.000
339	Analyze Purchase Requisition	2011/02/17 15:34:00.000	2011/02/17 15:40:00.000
339	Amend Purchase Requisition	2011/02/18 03:29:00.000	2011/02/18 03:52:00.000
339	Analyze Purchase Requisition	2011/02/18 23:24:00.000	2011/02/18 23:30:00.000
339	Create Request for Quotation Requester Manager	2011/02/18 23:36:00.000	2011/02/18 23:38:00.000
339	Analyze Request for Quotation	2011/02/22 15:34:00.000	2011/02/22 15:58:00.000
339	Amend Request for Quotation Requester	2011/02/22 16:50:00.000	2011/02/22 17:03:00.000
339	Analyze Request for Quotation	2011/02/28 14:10:00.000	2011/02/28 14:34:00.000
940	Create Purchase Requisition	2011/05/17 11:31:00.000	2011/05/17 12:08:00.000
940	Create Request for Quotation Requester	2011/05/17 14:58:00.000	2011/05/17 15:06:00.000

Figure 6: *database_ example*

2. **Process mining activity:** Now, once we have that data, process mining can really discover the actual process based on that data. So here you have a simple example scenario. You see different activities that perform for different cases. Assume each case is a computationally ordered process.
   Delivery process
   . Step A: the customer who orders our product
   . Step B: the customer paying the product
   . Step C: shipment of the product to the customer.

. Step D: check the correct address to deliver

. Step E: sign and deliver the product to the customer

Look customer 1 (below here): process activity is A -> B -> C -> D -> E

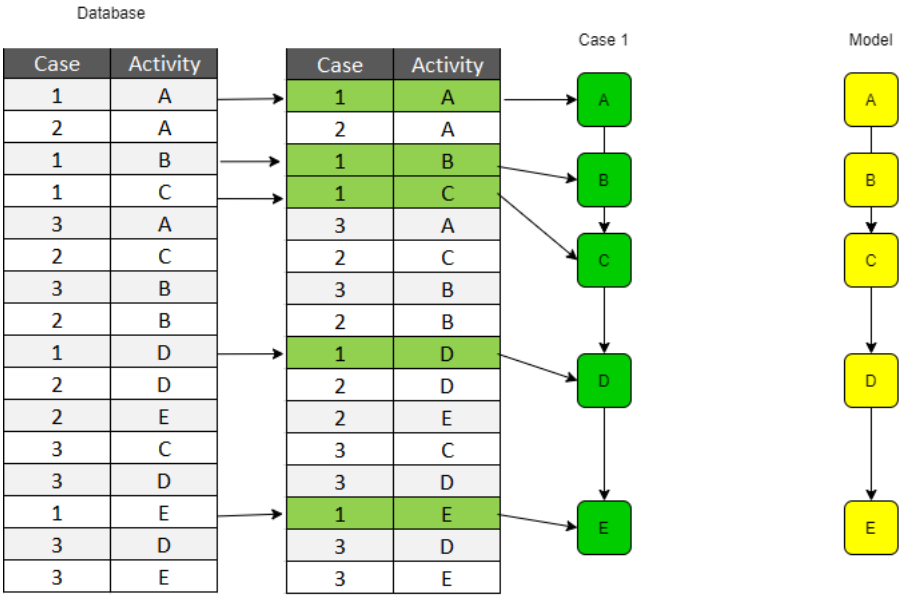


Figure 7: *database_ example*

Look customer 2 (below here): we can see that the process went through a similar flow. But it did not went along exactly the same path. If you look closely, you will see that B and C are happening in a different order here

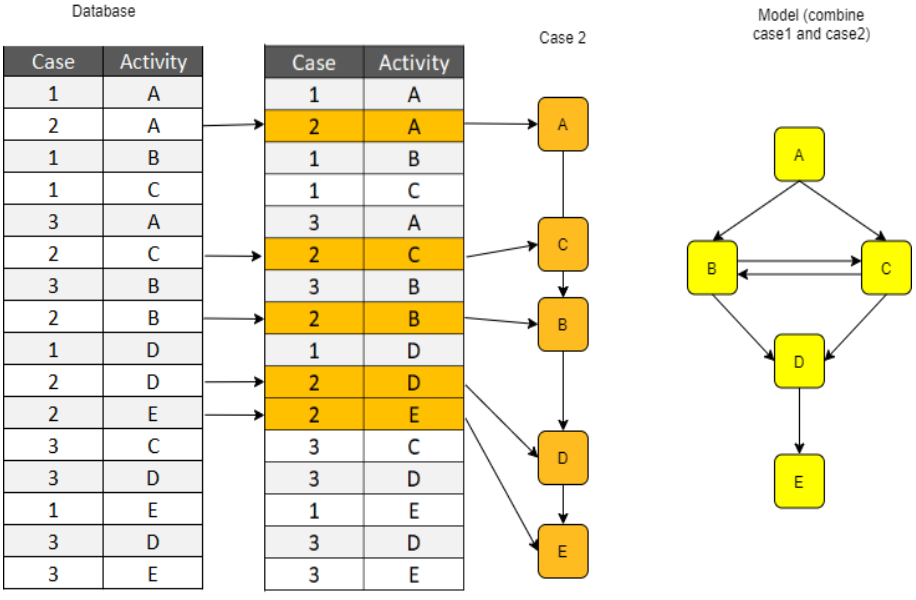


Figure 8: *database_ example*

Look at customer number 3 (below here): then we see there's a repetition around activity D

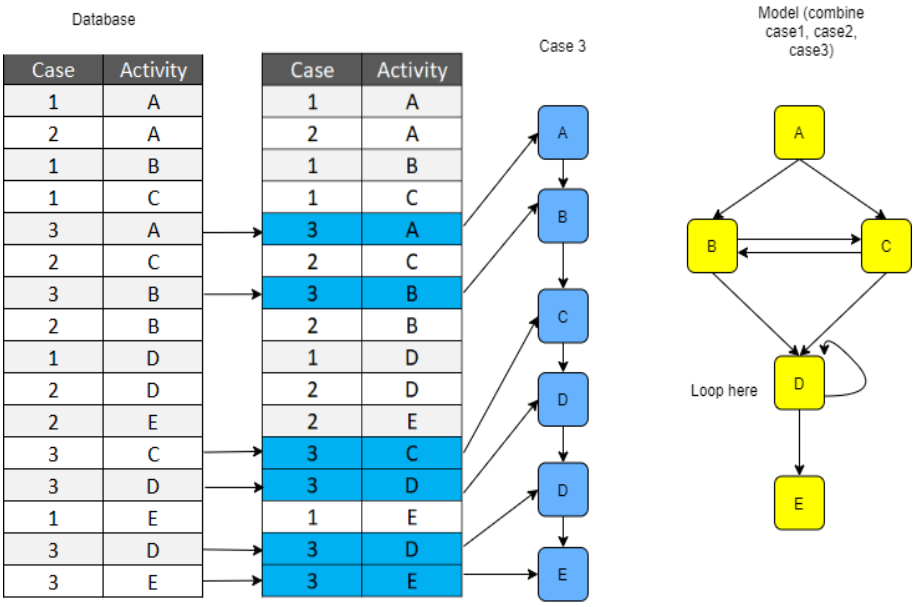


Figure 9: *database_ example*

6.2.3 Phase3: Data Analysis

+ Process Discovery and Simplification
+ Bottleneck Analysis
+ Compliance Analysis

6.2.4 Phase 4: Presentation

+ Keep Copies of your Analyses
+ Take Different Views on your Process
+ Exporting Results

6.3 DISCO - TOOL FOR PROCESS MINING:

6.3.1 About:

Disco stands for discovery, and disco is on the market now since 2012. The program really a great academic tool set, which makes it very easy for researchers to develop new algorithms, for example, Disco's really meant for business professional, who does not need to know anything about how these algorithms work, and how the technology looks like under the hood. We have hidden that away from them, but we make it available for them so that based on their domain knowledge, and process knowledge, they can use these results and draw the right conclusions and interpretations from it.

Figure 10: *database_ example*

6.3.2 Feature:

Automated process discovery Create beautiful and insightful process maps directly from your raw data, automatically. Pick your desired level of abstraction. Create filters directly from activities or paths

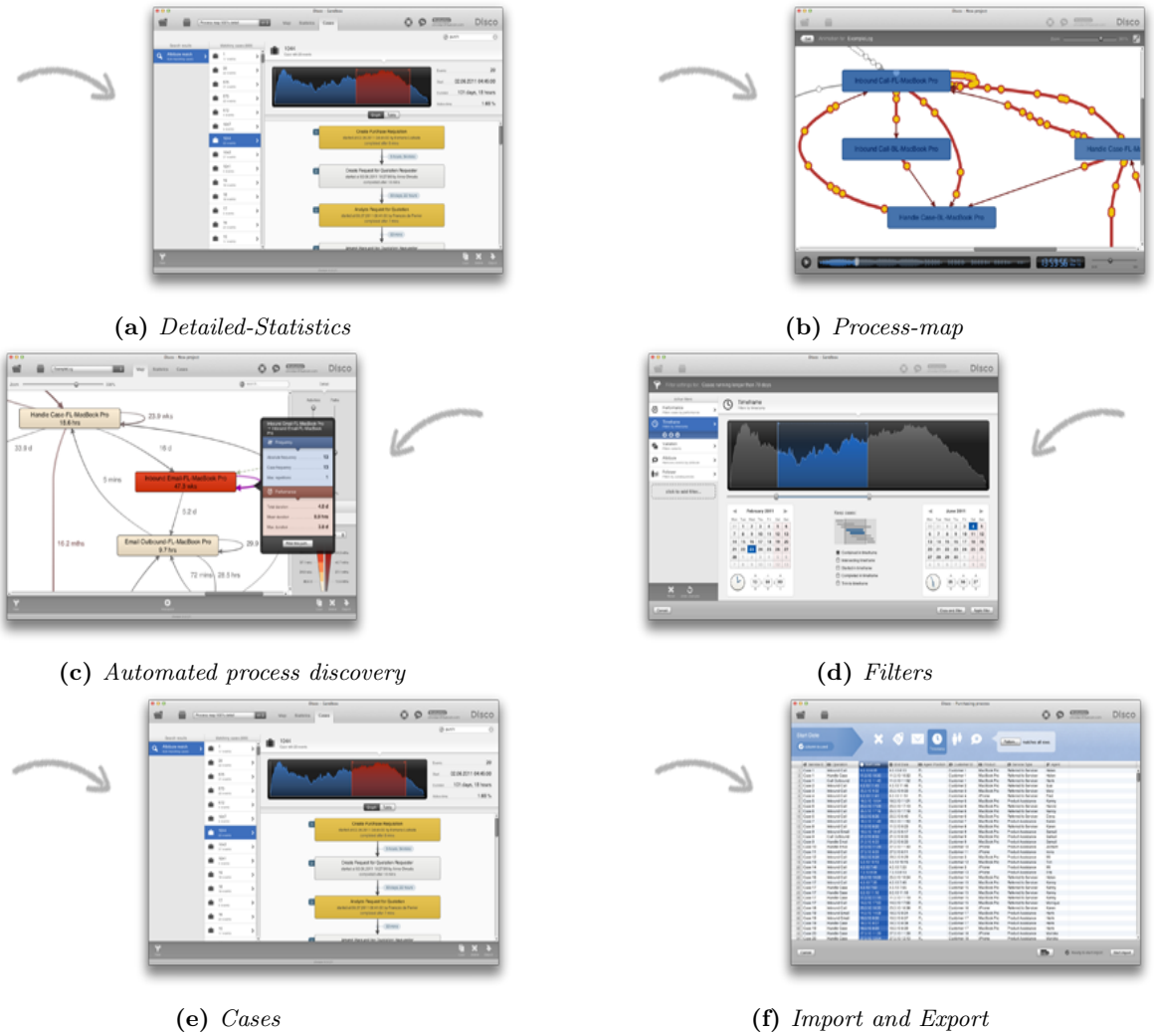
Process map animation: Create breathtaking animations, visualizing your process as it happened, right on your process map. Animation can help you to instantly spot bottlenecks where work is piling up.

Detailed Statistics: Get an overview about your data from sleek, interactive charts, and drill down into detailed information about each activity, resource, and attribute value

Cases: Quickly inspect the full history of the relevant cases, and find them with Disco's lightning-fast live search feature. You can see which cases follow the sunny day process path and which are exceptions.

Filters: makes it a snap to clean up your process data and to focus your analysis which includes: case performance, timeframe, variation, attributes, event relationships, or endpoints

Import and export: import complex CSV and MS Excel files. Read and write preconfigured data in ProM's MXML format, and in XES, the official IEEE-approved format for event log interchange.



(a) *Detailed-Statistics*

(b) *Process-map*

(c) *Automated process discovery*

(d) *Filters*

(e) *Cases*

(f) *Import and Export*

Figure 11: *Features of Disco*

6.4 PROCESS MINING ANALYSE

6.4.1 Understand Data

Data: PurchasingExample.fbt

	Case ID	Activity	Start Time	End Time	Resource	Role
1	339	Create Purchase Requisition	2011/02/16 20:31:00.000	2011/02/16 21:23:00.000	Nico Ojenbeer	Requester
2	339	Analyze Purchase Requisition	2011/02/17 15:34:00.000	2011/02/17 15:40:00.000	Maris Freeman	Requester Manager
3	339	Amend Purchase Requisition	2011/02/18 03:29:00.000	2011/02/18 03:52:00.000	Elvira Lores	Requester
4	339	Analyze Purchase Requisition	2011/02/18 23:24:00.000	2011/02/18 23:30:00.000	Heinz Gutschmidt	Requester Manager
5	339	Create Request for Quotation Requester Manager	2011/02/18 23:36:00.000	2011/02/18 23:38:00.000	Francis Odell	Requester Manager
6	339	Analyze Request for Quotation	2011/02/22 15:34:00.000	2011/02/22 15:58:00.000	Magdalena Predutta	Purchasing Agent
7	339	Amend Request for Quotation Requester	2011/02/22 16:50:00.000	2011/02/22 17:03:00.000	Penn Osterwalder	Requester Manager
8	339	Analyze Request for Quotation	2011/02/28 14:10:00.000	2011/02/28 14:34:00.000	Francois de Perrier	Purchasing Agent
9	940	Create Purchase Requisition	2011/05/17 11:31:00.000	2011/05/17 12:08:00.000	Immanuel Karagianni	Requester
10	940	Create Request for Quotation Requester	2011/05/17 14:58:00.000	2011/05/17 15:06:00.000	Esmana Liubiata	Requester
11	940	Analyze Request for Quotation	2011/05/19 00:30:00.000	2011/05/19 00:56:00.000	Francois de Perrier	Purchasing Agent
12	940	Send Request for Quotation to Supplier	2011/05/19 04:46:00.000	2011/05/19 04:59:00.000	Magdalena Predutta	Purchasing Agent
13	940	Create Quotation comparison Map	2011/05/19 08:44:00.000	2011/05/19 13:31:00.000	Francois de Perrier	Purchasing Agent
14	940	Analyze Quotation comparison Map	2011/05/19 20:38:00.000	2011/05/19 20:52:00.000	Kim Passa	Requester
15	940	Choose best option	2011/05/19 20:52:00.000	2011/05/19 20:52:00.000	Anna Kaufmann	Requester
16	940	Settle conditions with supplier	2011/05/21 04:31:00.000	2011/05/21 14:22:00.000	Magdalena Predutta	Purchasing Agent
17	940	Create Purchase Order	2011/05/21 23:48:00.000	2011/05/21 23:59:00.000	Francois de Perrier	Purchasing Agent
18	940	Confirm Purchase Order	2011/05/22 16:33:00.000	2011/05/22 16:44:00.000	Esmeralda Clay	Supplier
19	940	Deliver Goods Services	2011/05/23 10:32:00.000	2011/05/24 18:46:00.000	Esmeralda Clay	Supplier
20	940	Release Purchase Order	2011/05/25 01:59:00.000	2011/05/25 02:00:00.000	Kim Passa	Requester

Figure 12: *Data after import*

So this is the data that we've received about the purchasing process. Looking at the data we can see:

+ There are multiple steps that are being recorded for the same purchase order number. So for example, here we have eight steps that were performed for purchase order 339.

+ The second requirement was that we have information about the activities or steps or set of changes that happen in the process. So with process mining we need the whole history of all the steps that were performed in the past for all the cases.

+ We have not just one but even two timestamps. This is even better. One timestamp is the minimum requirement, but if you have two timestamps like here, this means that you will be able to analyze how long a specific activity actually took place. So how long someone was actively busy with a certain step in the process. And then you can isolate that from the waiting time that happens in the process as well, where nobody is really working on a specific case. We have some extra information as well.

6.4.2 Process Mining

By pressing the Import button, we immediately have a process map without any thinking or pre-drawing.

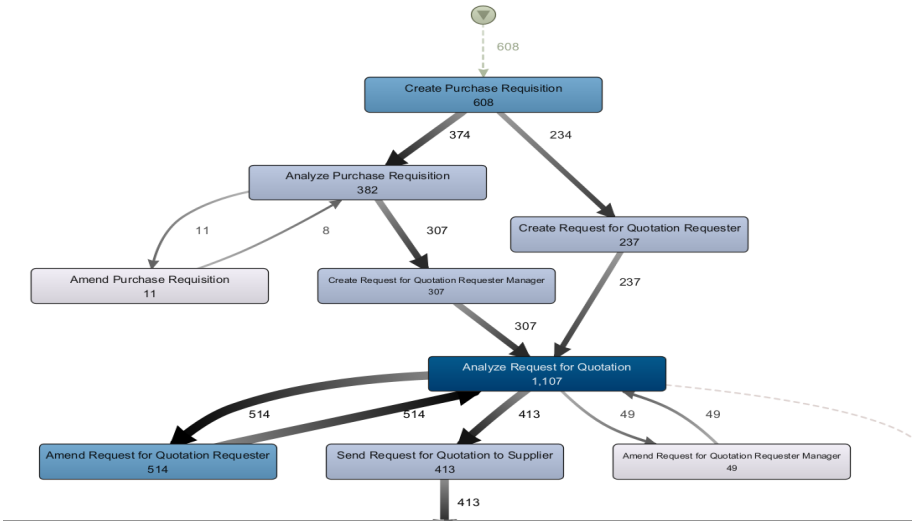


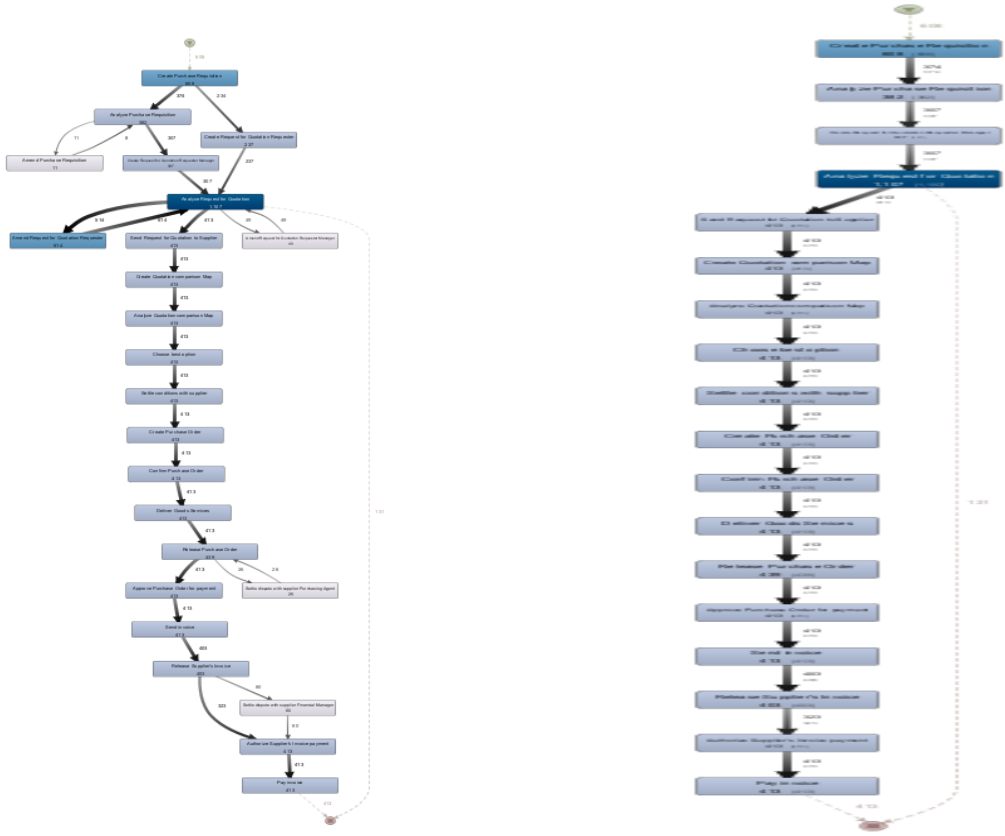
Figure 13: *Process-Map not full-size*

Read Process-Map

+ All of the 608 cases started with the step create purchase requisition. That's the first step in the process.

+ Afterwards, the process splits into two alternative path. 374 times you're going this way, 234 times you're going this way. For the numbers thickness of the arrows the coloring all reflect the frequency of how frequently certain parts of the process were performed. Now immediately we can see one problem here in this process which is a very dominant rework loop around an activity called iment request for quotation. It happens more than 500 times for just 608 cases, so that's an enormous waste that's going on here.

+ Now, one thing is really important once you start looking at the process based on data you will get to a very detailed level of the process. Because you will see every little exception. And everything that happens. And it can become very complex very quickly. Just because reality really is rather complicated. So you need to have ways to deal with that complexity. Through those sliders here that you can see on the screen's right hand, the map will become simpler.



(a) *original process map* **(b)** *process map has been simplified*

Figure 14: *pull down the activity slider to the lowest point.*

+ What you will see now is the process just based on the activities that are contained in the most frequent variant of the process. So this shows us the main flow of the process. And we can then gradually determine to bring it in in more detail. To reveal more activities that are less frequently performed by pulling up the slide more and more. Up to 100% in which case we see all of the steps that were ever recorded in the system. e can pull up the path lighter at 100% to now reveal all the path. So now we really see 100% of everything, every step that was performed in the process and all the transitions between those different activities.

Statistics, Cases and Variants: we had several questions about the process, and one of them was about the throughput time. So let's further explore the process and the data set a bit further.

+ We can then go to the statistics to see some more detailed statistics, some overview process statistics about the dataset.
  . We will see that there are 608 cases, 608 purchase orders in the data set. We already knew that from the process map.
  . There are 9,000 events so this is a relatively small data set. You can have millions of records in the data that can be analyzed with process mining.
  . We also see the time frame of the data that is being covered. For example, here we have January to October 2011. So that's about ten months of data.

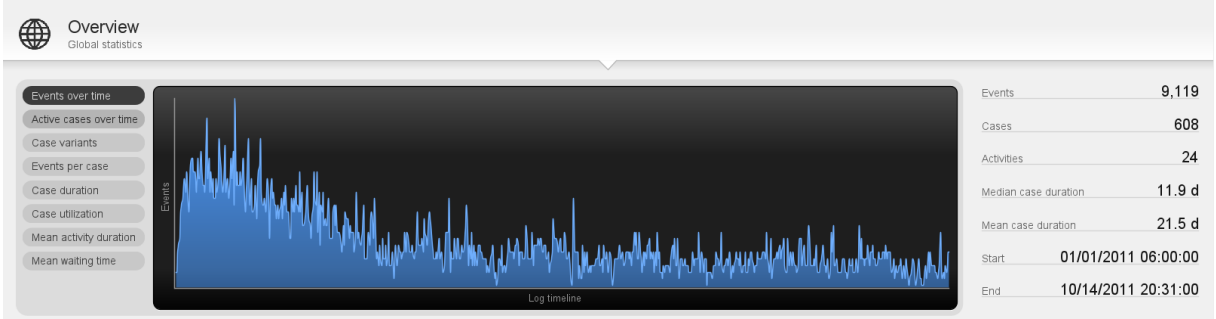


Figure 15: *Statistics about data*

+ On the left side we can now go to the case duration because we have those most cases are finished within 16 or 17 days from the very beginning to the very end of the process. So this is within our 21-day limit. However, we can see that on the right side, there are also quite a few that are taking really, really long, 80 days, 90 days and even more. And this doesn't seem to be an exception

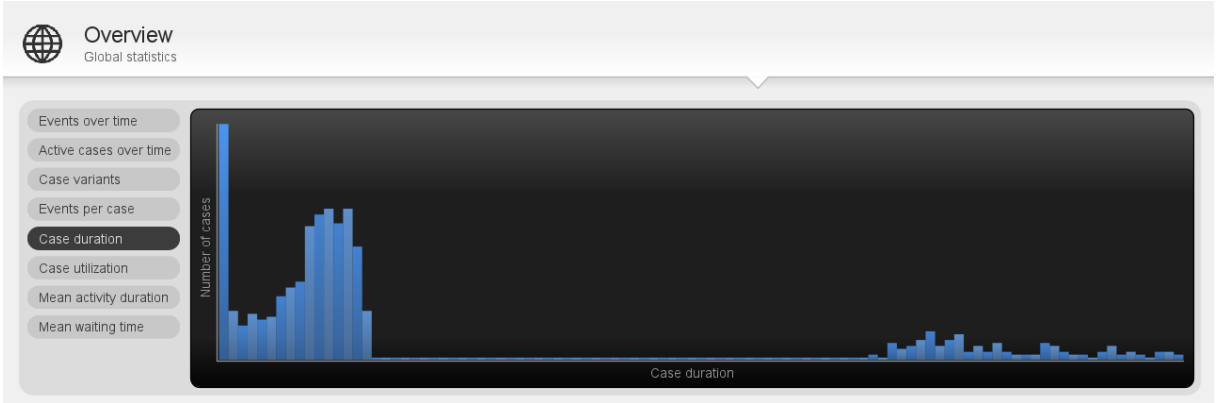
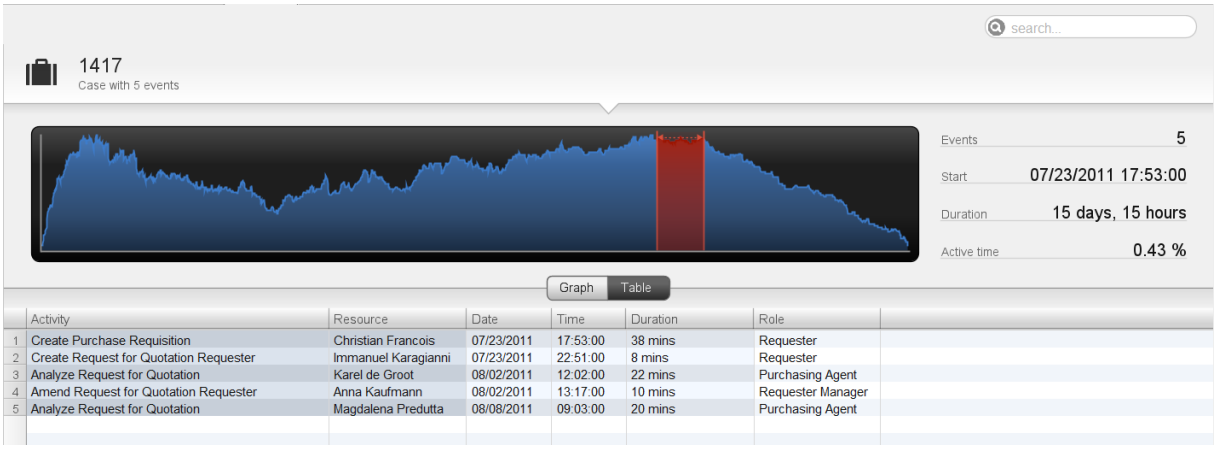


Figure 16: *Statistics about data*

+ We can go from the Statistics to the Cases view. And the Cases view shows us information about individual cases.

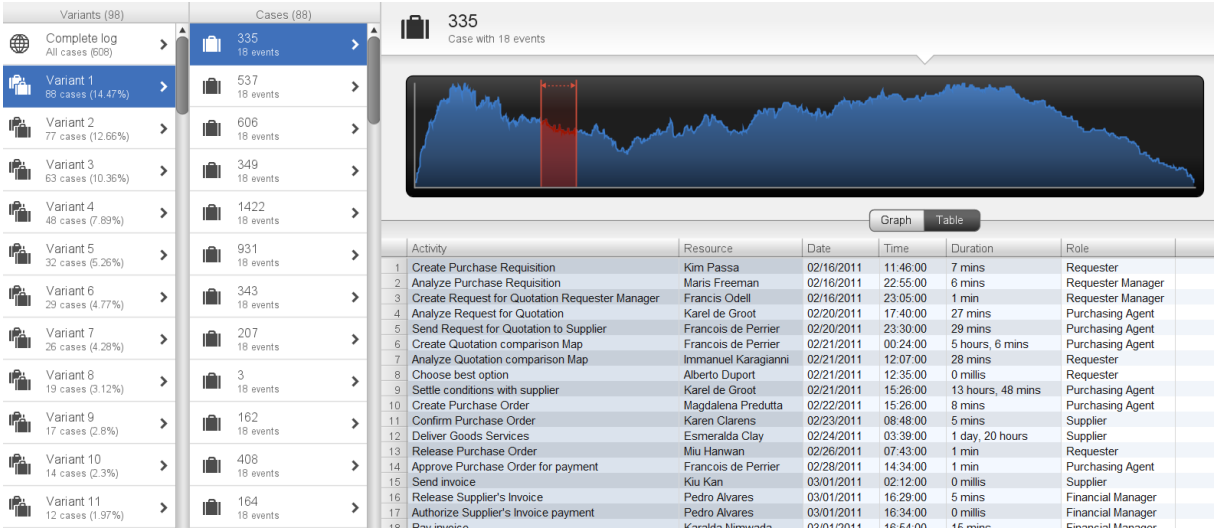


	Activity	Resource	Date	Time	Duration	Role
1	Create Purchase Requisition	Christian Francois	07/23/2011	17:53:00	38 mins	Requester
2	Create Request for Quotation Requester	Immanuel Karagianni	07/23/2011	22:51:00	8 mins	Requester
3	Analyze Request for Quotation	Karel de Groot	08/02/2011	12:02:00	22 mins	Purchasing Agent
4	Amend Request for Quotation Requester	Anna Kaufmann	08/02/2011	13:17:00	10 mins	Requester Manager
5	Analyze Request for Quotation	Magdalena Predutta	08/08/2011	09:03:00	20 mins	Purchasing Agent

Figure 17: *Case 1417 activity*

+ So this is purchase order 1417, and we see that these five steps were performed for that particular purchase order. And we see the people who were involved, the timestamps, and any additional attributes that we chose to import as well. But for any kind of problem, any things that you find in your analysis, you can always go back to a concrete example, to a concrete case to do a real root cause analysis, and find out what's going on.

+ And the second thing that you have in this Cases view is also a view of the different variants. So if we look here on the left, we can see first of all that there are 98 different variants in this data set, which consists of 608 cases. So that gives you a sense of the amount of variation that there is in the first place. But then as a next step, you can look at the top variants, maybe the top five or the top ten variants. And one variant here is really one sequence through the process from the very beginning to the very end of the process. And of course there can be multiple cases that follow exactly the same path through the process from beginning to end.



	Activity	Resource	Date	Time	Duration	Role
1	Create Purchase Requisition	Kim Passa	02/16/2011	11:46:00	7 mins	Requester
2	Analyze Purchase Requisition	Maris Freeman	02/16/2011	22:55:00	6 mins	Requester Manager
3	Create Request for Quotation Requester Manager	Francis Odell	02/16/2011	23:05:00	1 min	Requester Manager
4	Analyze Request for Quotation	Karel de Groot	02/20/2011	17:40:00	27 mins	Purchasing Agent
5	Send Request for Quotation to Supplier	Francois de Perrier	02/20/2011	23:30:00	29 mins	Purchasing Agent
6	Create Quotation comparison Map	Francois de Perrier	02/21/2011	00:24:00	5 hours, 6 mins	Purchasing Agent
7	Analyze Quotation comparison Map	Immanuel Karagianni	02/21/2011	12:07:00	28 mins	Requester
8	Choose best option	Alberto Duport	02/21/2011	12:35:00	0 millis	Requester
9	Settle conditions with supplier	Karel de Groot	02/21/2011	15:26:00	13 hours, 48 mins	Purchasing Agent
10	Create Purchase Order	Magdalena Predutta	02/22/2011	15:26:00	8 mins	Purchasing Agent
11	Confirm Purchase Order	Karen Clarens	02/23/2011	08:48:00	5 mins	Supplier
12	Deliver Goods Services	Esmeralda Clay	02/24/2011	03:39:00	1 day, 20 hours	Supplier
13	Release Purchase Order	Miu Hanwan	02/26/2011	07:43:00	1 min	Requester
14	Approve Purchase Order for payment	Francois de Perrier	02/28/2011	14:34:00	1 min	Purchasing Agent
15	Send invoice	Kiu Kan	03/01/2011	02:12:00	0 millis	Supplier
16	Release Supplier's Invoice	Pedro Alvares	03/01/2011	16:29:00	5 mins	Financial Manager
17	Authorize Supplier's Invoice payment	Pedro Alvares	03/01/2011	16:34:00	0 millis	Financial Manager

Figure 18: *Varient 1 - Case 335*

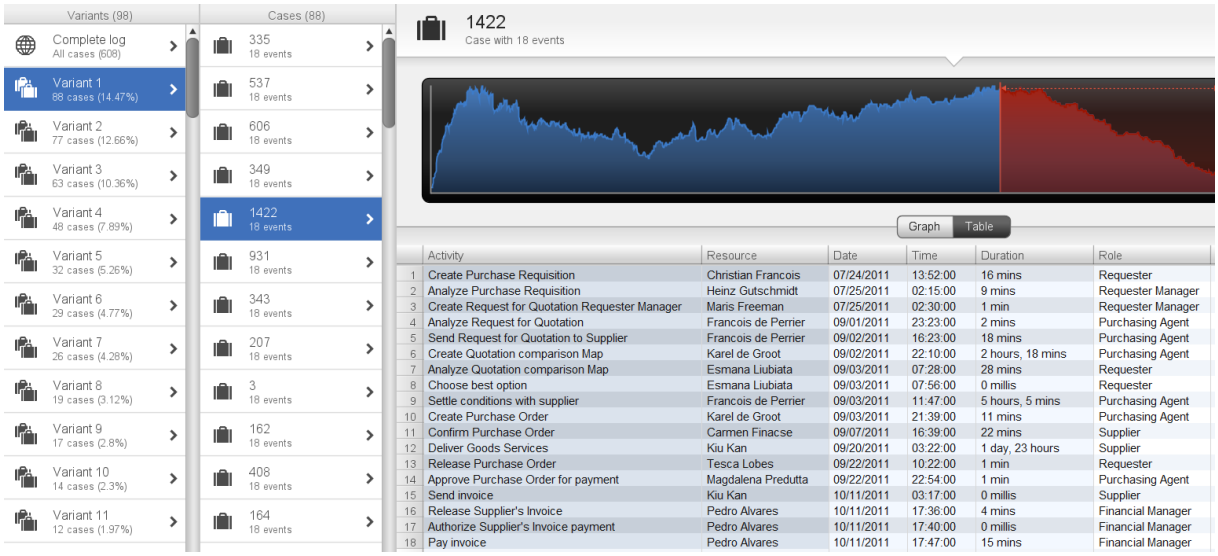


Figure 19: *Varient 1 - Case 1422*

+ Variants are sorted in descending order of cases.So, if we look at variant 1, we see that this is covering around 15% of all the instances in our data set. And we see here those 88 cases, they all follow the same variant. So, you won't see any changes in the sequence of steps because they all follow the same path. But you will see different people involved, different time stamps and so on. So by stepping through the different variants, we can often get a good understanding already of the maybe 60, 70% of the process, by looking at the most frequent variants.

+ If we look at variant 3 in our purchasing process, this is rather interesting as well actually because we see that the third most frequent variant is one where the process is stopped quite early. So in our case we only extract the completed purchasing request. So there's no cases that are still in progress, which is another topic that's important for process mining which we will come back to later. But in this case we only have completed purchase orders. What we can see though, is that after the analyze purchase requisition step, the purchase was stopped.

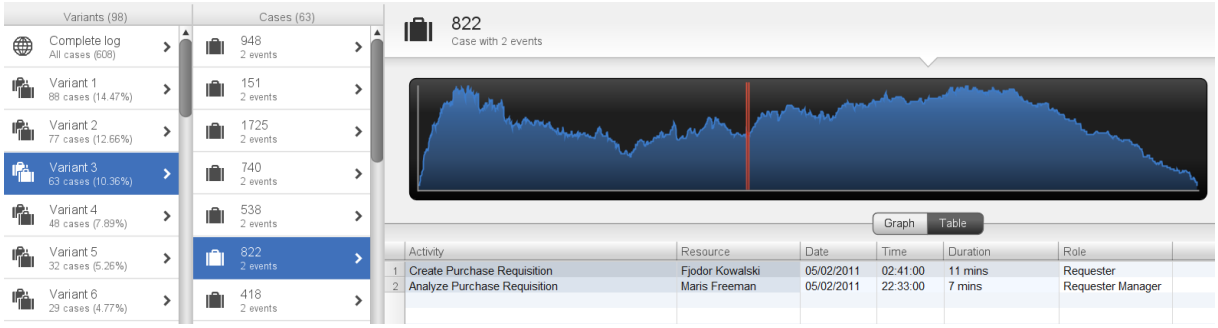


Figure 20: *Varient 2 - Case 822*

+ So one reason might be that this purchase wasn't approved, and yeah, it shouldn't be in system in the first place. So you could consider those cases in variant 3 another form of waste because these two steps could have been prevented if we would have not started the case in the first place. And that's another example of insights that you can draw with process mining, but we can also see that if we go back to the process map. You can see this here with this dashed line showing one of those early end points. We can see it's leaving the process here after the second step. And if we scroll towards the end

of the process, then we will see that there's the main process flow here up till the very end. And while we have just one start point in the process, we actually have multiple end points.

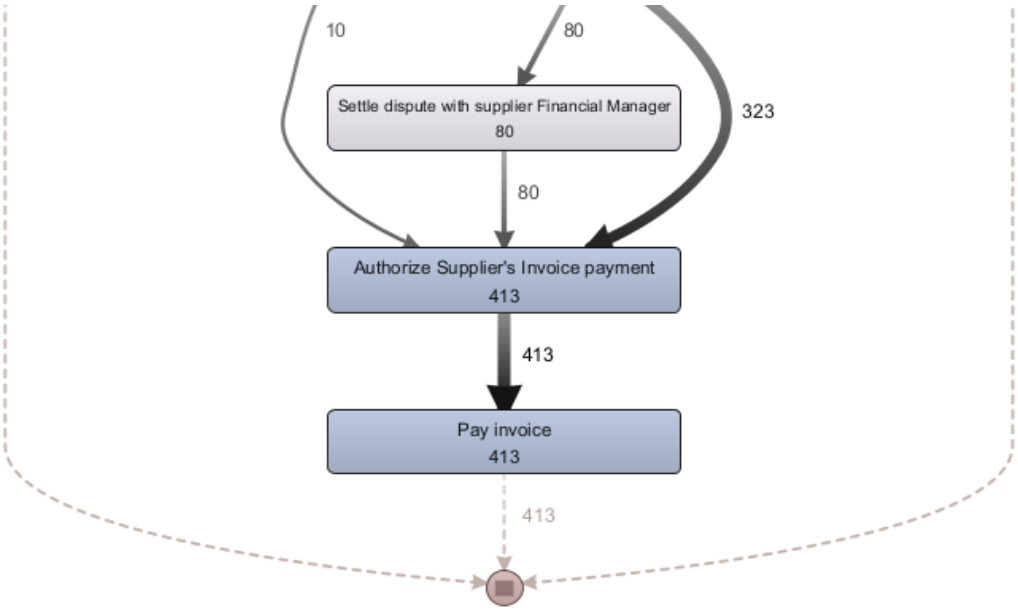


Figure 21: *Multiple End Points*

=> We have constructed a complete picture of how the processes really performed in the process discovery, and we found already a couple of problems with this process. We saw that there are a lot of amendments. So these were the loops that we found in the process map where there's a lot of reworks through changes that are made to existing requests. And in the cases that we found that they are a lot of stopped requests after the second step in the process in this third variant that we discovered. So these are improvement opportunities for our process. And we can now start thinking about how we can prevent those rework loops and how we can prevent those unnecessary cases. There are quite a few cases that are taking longer than 21 days. So this is what we are going to analyze now as a next step.

Bottleneck Analysis

+ So to find out where we are losing so much time in the process, if we are taking so long in total, we are using a filter, in this case this is a performance data that we are using. So what we see here is the overall duration, a histogram for those process durations and we see that there are those really long running ones here on the right side and we want to focus on those. But, let's just take the 21 day limit that we have as a target value for our system for our process and we use that as the border. So only the blue cases, the ones that are taking longer than 21 days, these are the problematic ones, so we are going to focus on them in the subsequent analysis.

+ We can already see that roughly 15% of all the cases fall into that category of problematic long running cases. Now once we apply the filter, we will see a little reminder here in the lower left corner that we are not looking at the full data set at the moment, but these are just the 15%. So just the 92 out of the 608 cases that are taking longer than 21 days. Only for those, we now see the process map shown.

+ And first of all we can see that this re-work loop (Amend request for quotation) is even more dominant than before. We are now going through this loop almost three times per case, right? So that certainly has something to do with those long case duration. But at the same time, we are not that interested in the frequency here. But we want to see where are the waiting times and the bottlenecks in the process. But to do that, we can switch from the frequency view on the right side to the performance view. And in the performance view let's go to the average duration.

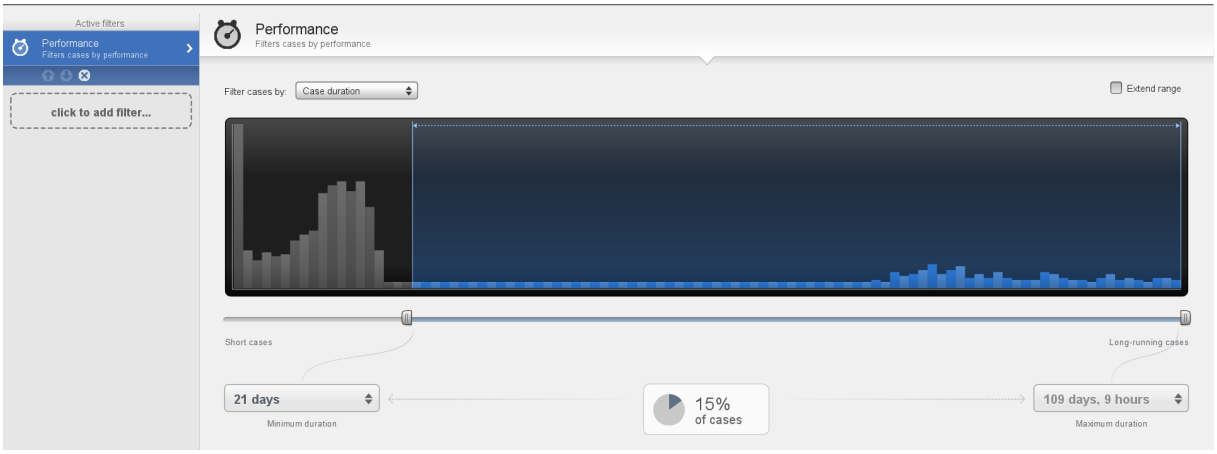


Figure 22: *Multiple End Points*

+ And what we can see here is that not only are we unnecessarily often going through this rework loop that we discovered earlier, but also the step itself doesn't take that long. This is about ten minutes on average that we are spending actively performing that particular step. But then afterwards, after this step is finished, there is on the average, well more than 14 days waiting time, idle time in the process where nothing is happening before the normal start of the process is picking up again. And we also see those delays emerging from other parts in the process all pointing to this particular activity and Analyze Request for Quotation, which really seems to be our bottleneck here in this particular process.

+ There could be different root causes for that. For example, maybe there are not enough resources available for that particular step and they have a lot of workload, so that's why there are delays. It could also be that this is a low priority task being performed by a manager every four weeks and then in the meantime, things are piling up. Processes are something rather abstract and people often have problems with understanding, thinking and processes. So this visualization can really help you a lot in communicating the problems that you found in your analysis results. And in the same line I want to show you the animation as well. So we have found this bottleneck but we can even bring it to life by using the animation.

+ We are still focusing on those 15%, on those slow, long-running cases here. But now we don't see an average view with average duration. But we see a dynamic view, which is a replay of the actual process over time. So every yellow dot is one case. This is one purchase order moving through the system. And, keep in mind that this is not a simulation like you might know it from process modeling tools. But this is a replay of the actual process. The timestamps are being taken from the actual process executions. And we can see the actual replay of the whole process over the course of these ten months that we have in the data. And if we move forward a little bit in the timeline, then we can clearly see those bottlenecks emerge, right? You can really see how things are queuing up and where they're piling up. And what this does is, it makes it really tangible for people to understand where the problem is.

  => We know that there are performance problems, there a lot of cases taking more than 21 days, 15% of all cases take longer than 21 days in fact. And we can also see now, where the bottleneck is. The Analyze Request for Quotation activity is the food cost for those delays that are taking place. And we have to address that problem and we have to change the process in a certain way depending on the actual road cost that we have to find out with further investigation to resolve that bottleneck.

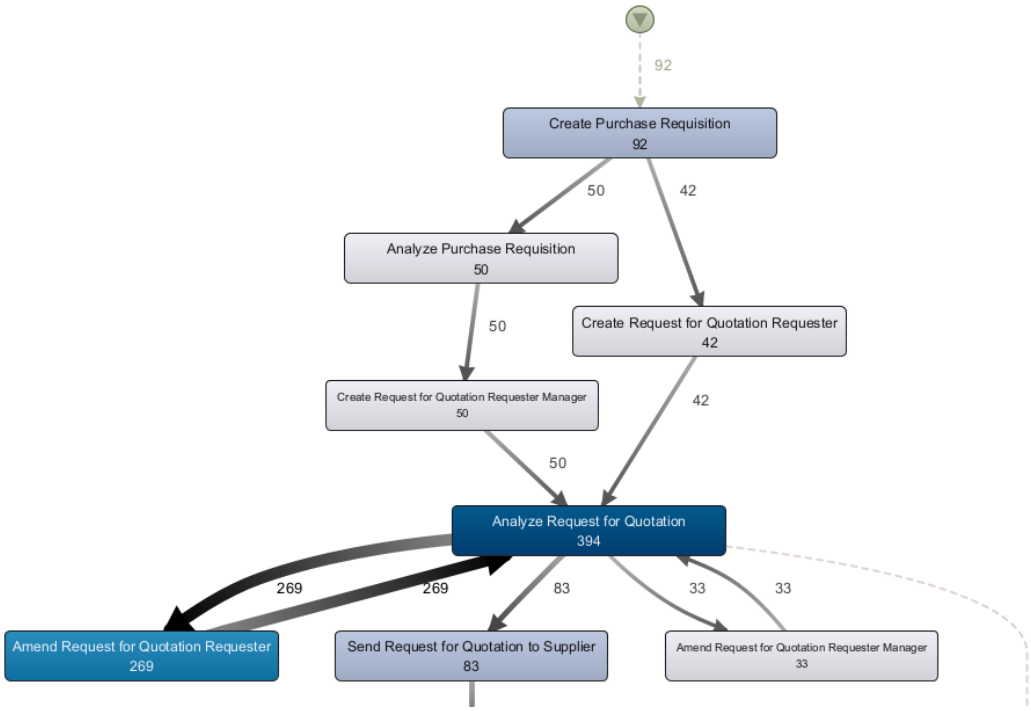


Figure 23: *Process Map after use filter*

Compliance Analysis

+ If we inspect the process carefully then we will find one compliance problem in this process flow that we did not expect. There is one mandatory activity in this process which is called release suppliers invoice which is shown here. So every purchase order that is being sent out as an invoice ultimately needs to be explicitly released and this activity is there to prevent fraud so this is part of our purchasing guidelines that we need to adhere to.

+ What we can see is that there are actually ten cases that are bypassing this mandatory step. So they're sneaking along here and jumping right to the authorized supplier invoice payment part. And this is a problem. Maybe we didn't even know that it was possible in the system to do that, but somehow people have found a way around it., and we can see not only that it happens, but also, how frequently it happens. So ten times this step was skipped.

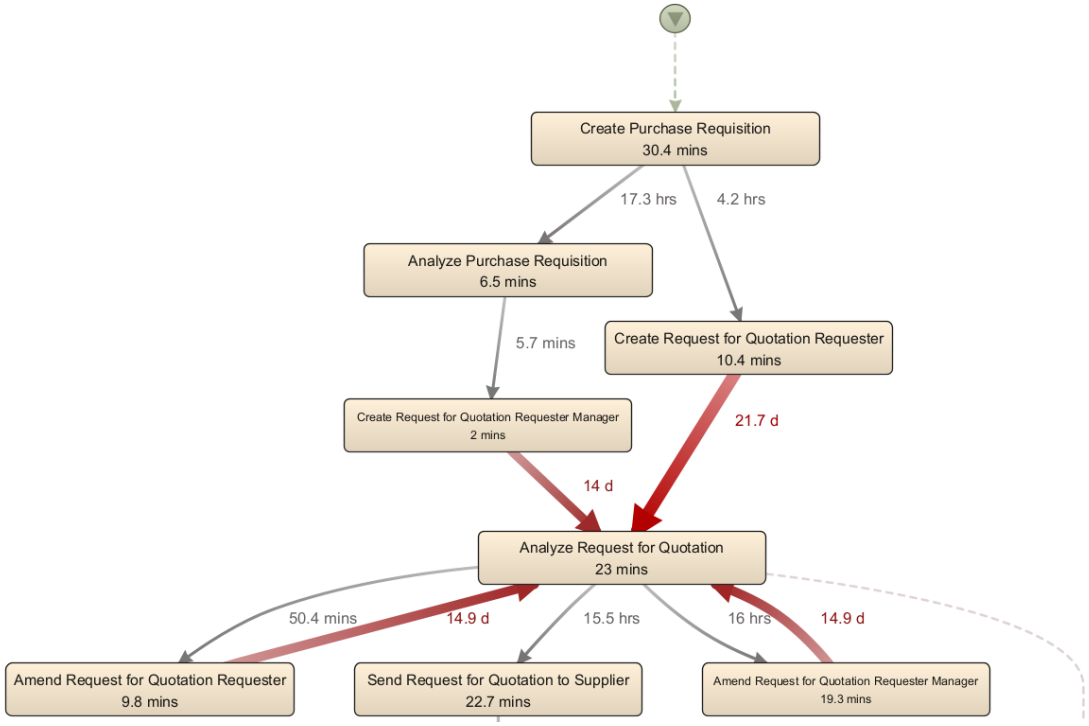


Figure 24: *Process Map after use performance view*

+ Next step we want to know which of the ten cases are this. So what I can do to find that out is I click on this arrow and then I have this little button here which says filter this path. This is a shortcut that gives me information about all the cases that are following this particular path in the process.

+ I get the process map for those ten cases, which in this case, also not terribly interested in. I want to see the cases themselves. So I would go to the cases view, and here I have now a list of those ten cases. Maybe some exception to the rule that I forgot. So that could be one reason. If that's not the case, and they actually should have performed this step.

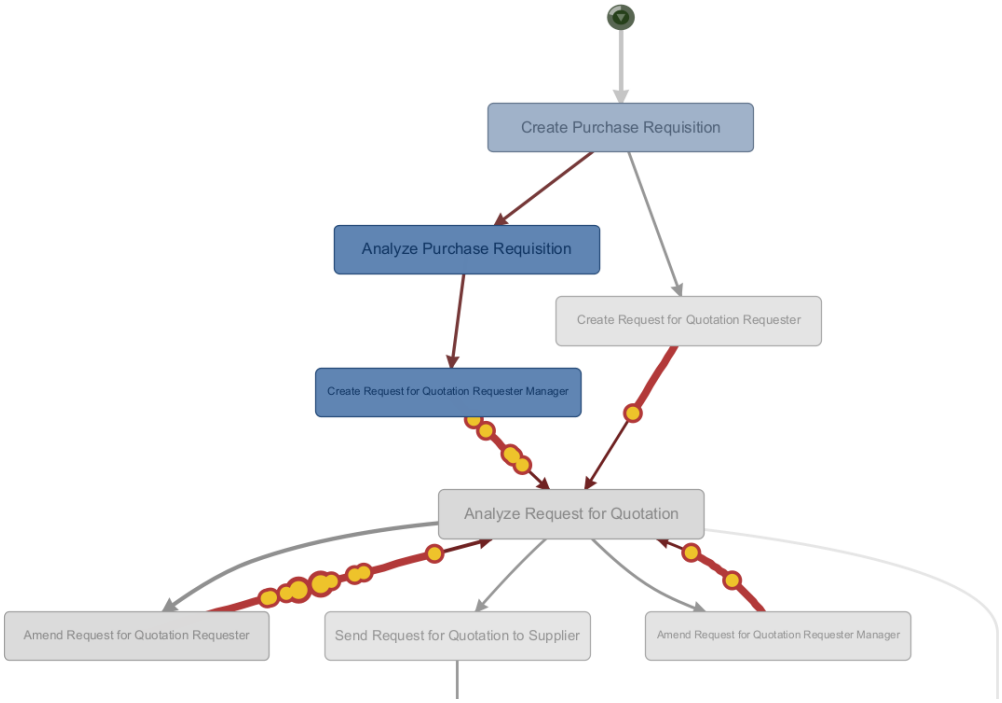


Figure 25: *Animation view*

6.4.3 Result:

(a) **How does the process actually look ?**

+ Objective process map discovered
+ Lots of amendments and stopped request: Update or purchasing guidelines needed

(b) **Do we meet the performance targets ?**

+ Not by all (some take longer than 21 days)
+ The "Analyze Request for Quotation" activity is a huge bottleneck: Process change is needed here

(c) **Are there deviations from the prescribed process?** There are 10 proven cases above that do not follow the rules => Training or system change needed.

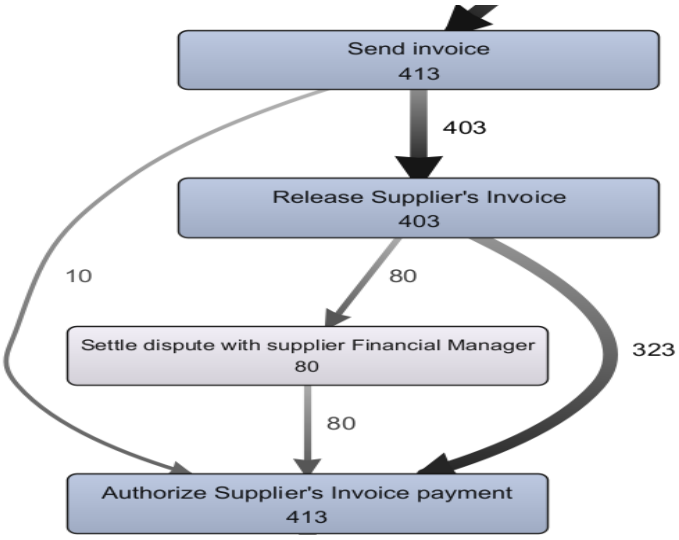


Figure 26: *Non-compliance at the send invoice step*

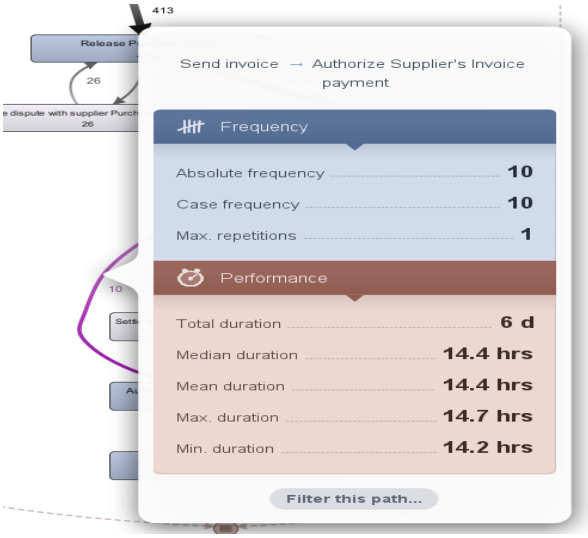


Figure 27: *Analyze non-compliance*

7 REFERENCES

- RaphaëlJs Tutorial - Sebastián Gurin
- Modeling Business Processes - A Petri Net-Oriented Approach - Wil M.P. van der Aalst, Christian Stahl
- MATHEMATICAL MODELING And RISK ANALYSIS
- FROM PETRI NETS TO COLORED PETRI NETS: A TUTORIAL INTRODUCTION TO NETS BASED FORMALISM FOR MODELING AND SIMULATION - Vijay Gehlot
- Process Mining: Data science in Action
- Process Mining and Automated Process Discovery Software for Professionals - Fluxicon Disco.
- evondev Youtube Channel

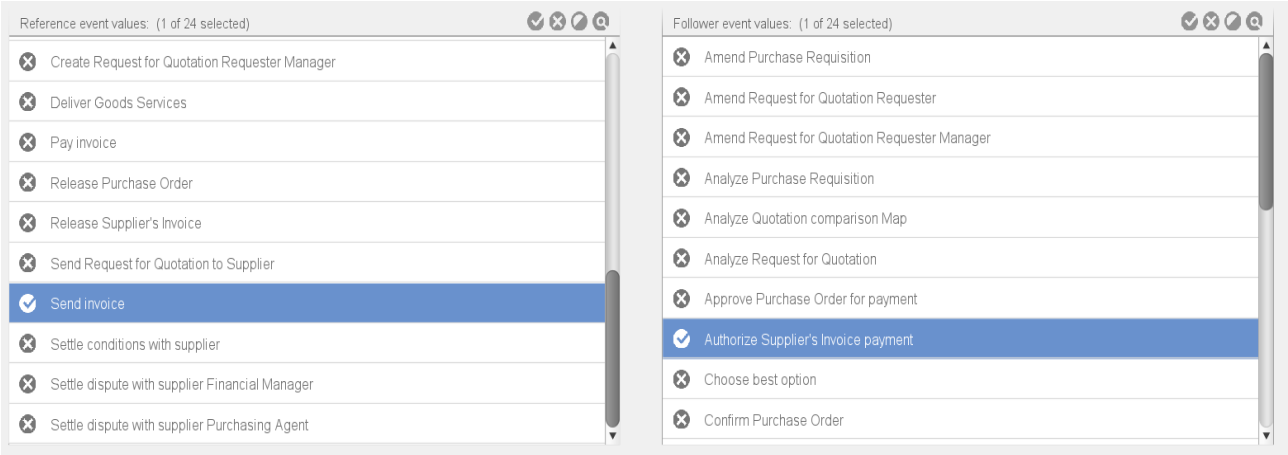


Figure 28: *After click filter button*

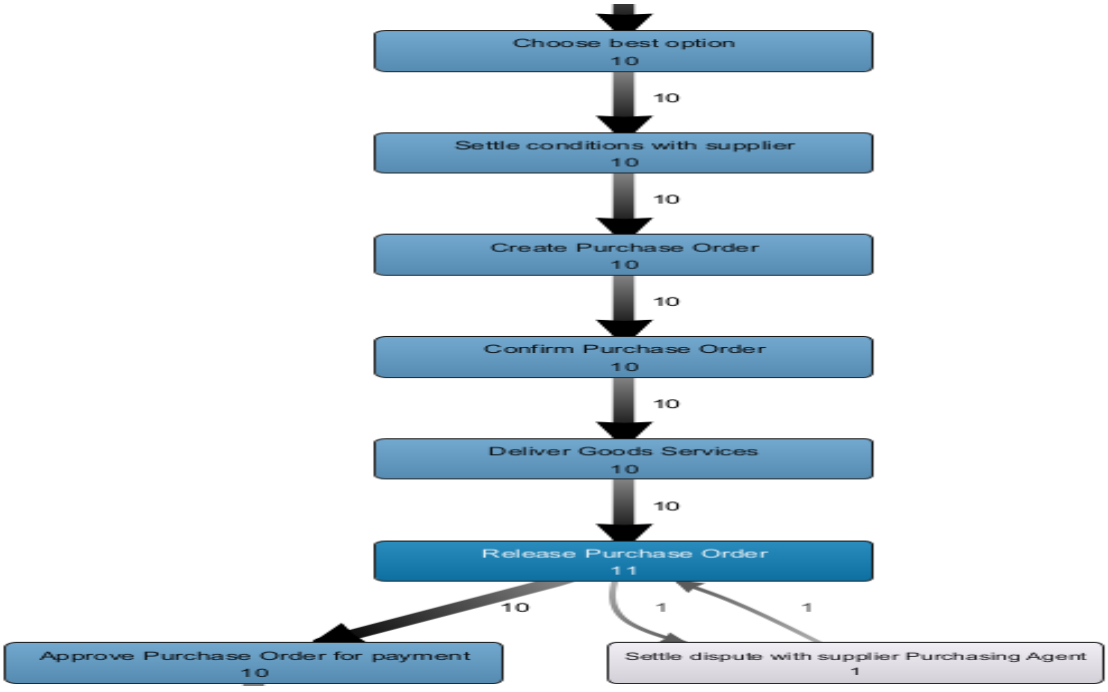
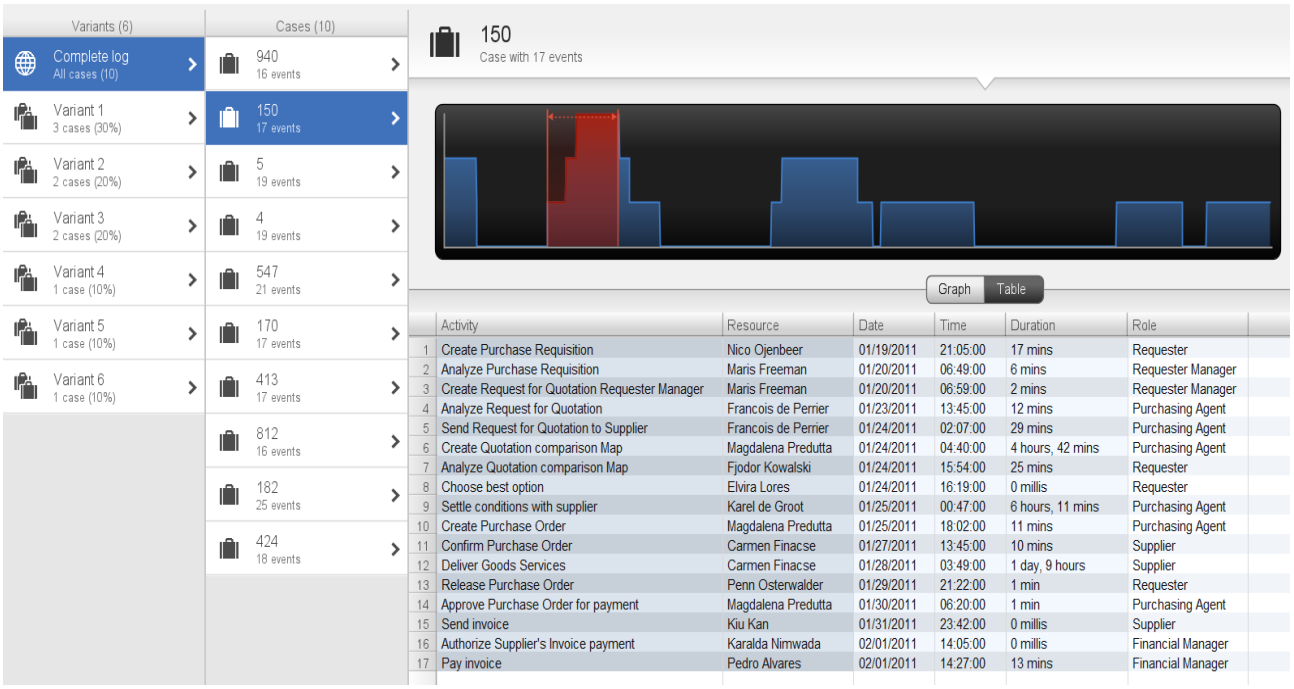


Figure 29: *process map after click filter button*



	Activity	Resource	Date	Time	Duration	Role
1	Create Purchase Requisition	Nico Ojenbeer	01/19/2011	21:05:00	17 mins	Requester
2	Analyze Purchase Requisition	Maris Freeman	01/20/2011	06:49:00	6 mins	Requester Manager
3	Create Request for Quotation Requester Manager	Maris Freeman	01/20/2011	06:59:00	2 mins	Requester Manager
4	Analyze Request for Quotation	Francois de Perrier	01/23/2011	13:45:00	12 mins	Purchasing Agent
5	Send Request for Quotation to Supplier	Francois de Perrier	01/24/2011	02:07:00	29 mins	Purchasing Agent
6	Create Quotation comparison Map	Magdalena Predutta	01/24/2011	04:40:00	4 hours, 42 mins	Purchasing Agent
7	Analyze Quotation comparison Map	Fjodor Kowalski	01/24/2011	15:54:00	25 mins	Requester
8	Choose best option	Elvira Lores	01/24/2011	16:19:00	0 millis	Requester
9	Settle conditions with supplier	Karel de Groot	01/25/2011	00:47:00	6 hours, 11 mins	Purchasing Agent
10	Create Purchase Order	Magdalena Predutta	01/25/2011	18:02:00	11 mins	Purchasing Agent
11	Confirm Purchase Order	Carmen Finacse	01/27/2011	13:45:00	10 mins	Supplier
12	Deliver Goods Services	Carmen Finacse	01/28/2011	03:49:00	1 day, 9 hours	Supplier
13	Release Purchase Order	Penn Osterwalder	01/29/2011	21:22:00	1 min	Requester
14	Approve Purchase Order for payment	Magdalena Predutta	01/30/2011	06:20:00	1 min	Purchasing Agent
15	Send invoice	Kiu Kan	01/31/2011	23:42:00	0 millis	Supplier
16	Authorize Supplier's Invoice payment	Karalda Nimwada	02/01/2011	14:05:00	0 millis	Financial Manager
17	Pay invoice	Pedro Alvares	02/01/2011	14:27:00	13 mins	Financial Manager

Figure 30: *process map after click filter button*