This workflow implements a four-stage data processing pipeline. Sample raw data are provided in folder A.

1. **Stage 1 – Maintenance & Diagnostics Integration**: Merges corrective maintenance data, train composition, and diagnostic logs to find the **first occurrence time (in hours)** of each alarm after maintenance.
2. **Stage 2 – Alarm Statistics Estimation**: Processes these results to **estimate the average alarm rate (λ)** for each alarm type (folder B)
3. **Stage 3 – Fault Modeling in FaultFlow:**  
   Use the estimated λ values to **parameterize fault activation distributions** in FaultFlow for model-driven dependability evaluation (folder C)
4. **Stage 4 –** **librbd:**

Train reliability evaluation using the redundancy constraints provided for the given train configuration and the output of FaultFlow. (folder D)

**Input Files (Stage 1)**

**1. Maintenance File**

* File: maintenance.xlsx
* Structure:
  + One sheet per train (e.g., TSR 008)
  + Relevant columns:
    - Fine guasto: End date of the maintenance event
    - Sede Tecnica: Identifies the affected coach
    - Descr. assem. Padre: Only rows with value Trazione are processed

**2. Diagnostic Event Logs**

* Folder: Dati diagnostici
* Format: One CSV per train (e.g., TSR 008.csv)
* Relevant columns:
  + ts: Unix timestamp (milliseconds), converted to datetime
  + depot, event\_type, machine\_type, alert\_type, cod: used for filtering (see below)
  + source and name: used to associate events with specific coaches
  + id: diagnostic alarm ID

**3. Train Composition File**

* File: composizione\_treni.xlsx
* Structure:
  + One sheet per train
  + First row: real coach names
  + Header: coach positions (e.g., first, second, ..., last)
  + The **first coach name** is used for filtering source in diagnostic logs

**Processing Workflow**

**1. Maintenance records preprocessing**

* Only rows where Descr. assem. Padre == 'Trazione' are considered
* A maintenance\_time column is computed in order to take in consideration the next day at 00:00 after the Fine guasto date:
  + Set to **00:00 of the day after** the Fine guasto date

**2. Diagnostic Data Preprocessing**

* The diagnostic CSV is loaded and ts is converted to datetime
* Filters applied (we took in consideration only the alarms of the coaches that):
  + depot == 0
  + event\_type == "ON"
  + machine\_type == "MD"
  + alert\_type == "PDM"
  + cod == 5

**3. Integration with Train Composition**

* Each train's composition sheet is read to get the list of coach names
* Filtering logic:
  + For the **first coach**, events must match both source and name
  + For **other coaches**, only name must match
* maintenance data is further filtered to match rows where Sede Tecnica equals the coach identifier

**4. Alarm Timing Computation**

* For each maintenance event:
  + Define an interval: from current maintenance\_time to the next one (if any)
  + For each allowed alarm ID {1, 2, 3, 4, 5, 6, 7, 8, 9, 11}:
    - Find the **first event** in the diagnostic log that occurs in this interval
    - Compute the time difference (in hours) between the event and maintenance\_time

**Output**

* For each train, an Excel file is generated in the results folder:
  + Filename: e.g., TSR 008\_prima\_occorrenza.xlsx
  + One sheet per coach
  + Each sheet contains:
    - Fine guasto and maintenance\_time for that coach
    - One column per allowed alarm, showing the delay (in hours) to first occurrence after maintenance

**Stage 2 – Alarm Rate Estimation (λ)**

This second stage reads the Excel files produced in Stage 1 and calculates, **for each alarm type**, the estimated **rate of occurrence λ** using the formula:

λ=N\T

Where:

* N = number of observed alarms
* T = total observed time (sum of time intervals in hours)

**Optional Preprocessing**

If an alarm is missing in a row (NaN), and another maintenance\_time follows, it fills the value with the time difference between current and next maintenance. This assumes the alarm didn’t occur in that interval but is censored.

**Workflow**

1. Reads all Excel files in the folder risultati\_prima\_ocorrenzza\_PDM
2. For each coach sheet, applies optional preprocessing (ask via input prompt)
3. Combines all coach data
4. Estimates λ for each alarm type
5. Exports a CSV:
   * With or without preprocessing:
     + calculate\_statisctiche\_prima\_ocorrenzza\_PDM\_preprocess\_threshold.csv
     + calculate\_statisctiche\_prima\_ocorrenzza\_PDM\_threshold.csv

**Expected Folder/File:**

* maintenance file: maintenance.xlsx
* Diagnostic CSVs: in folder Dati diagnostici (giugno 2024 - con colonna pantografo)
* Composition file: composizione\_treni.xlsx
* The results folder is created automatically if it doesn't exist

**Stage 3 – Fault Modeling with FaultFlow (TCU System)**

After estimating alarm occurrence rates (λ values) in Stage 2, they are **manually integrated** into a FaultFlow model of the **Train Control Unit (TCU)**.

**Files Used**

**TCUSystemBuilder.java**

This class **defines the system model**, including:

* Components and their structure
* Faults and their failure logic
* Injection of λ values to parameterize fault occurrence distributions

Each λ value corresponds to a specific alarm type (e.g., alarm ID 7) and is assigned to a fault as follows:

double lambda7 = 0.0025; // example value from Stage 2 CSV

Fault alarm7 = new Fault("Alarm7");

alarm7.setOccurrence(new Exponential(1.0 / lambda7));

This configuration models **alarm 7 as an exponentially-distributed fault** with mean time to activation of 1/λ hours.  
The process is repeated for all relevant alarms (e.g., 1, 2, 3, 4, 5, 6, 7, 8, 9, 11).

**AnalysisLauncher.java**

This file **runs the evaluation** using the FaultFlow engine. It performs the following:

1. Instantiates the system using TCUSystemBuilder:

Component system = new TCUSystemBuilder().build();

1. Runs fault analysis:

FaultFlow.runAnalysis(system, "output/failure\_cdf.csv");

1. Exports results:
   * Failure duration distributions (.csv)
   * Importance measures (Birnbaum, Fussell-Vesely)
   * Execution time
   * Cut sets (if enabled)

**FaultFlow Output**

After executing AnalysisLauncher, the following files are typically generated:

* failure\_cdf.csv: time-to-failure distribution for key failure events
* birnbaum.csv, fussell\_vesely.csv: importance measures for faults
* cut\_sets.txt: minimal cut sets (optional)
* execution\_time.txt: total simulation duration

**Requirements**

* Python 3
* Python libraries:
  + pandas
  + openpyxl
* **Java 11+** and **Eclipse** for FaultFlow (stage 3)

**How to Run**

**1. Run Stage 1: Maintenance-Diagnostics Integration**

python script\_prima\_occorrenza\_allarme.py

Make sure the following files/folders exist:

* Avvisi SAP - Interventi manutentivi (Agg. giugno 2024).xlsx
* composizione\_treni.xlsx
* Dati diagnostici (giugno 2024 - con colonna pantografo)/

Results go to: results/

**2. Run Stage 2: Lambda Estimation**

python calculate\_lambdas.py

At runtime, it will ask:

Do you want to preprocess the diagnosted files? 1-yes, 2-no

Results can go to:

* calculate\_statisctiche\_prima\_ocorrenzza\_PDM\_threshold.csv
* calculate\_statisctiche\_prima\_ocorrenzza\_PDM\_preprocess\_threshold.csv

**3. Run Stage 3:**

**Ensure you have:**

* Java 11+
* Maven
* λ values correctly set in TCUSystemBuilder.java (copied from Stage 2 output)

**1. Build the project (from FaultFlow root):**

mvn clean install

**2. Run the analysis:**

faultflow/src/main/java/it/unifi/stlab/faultflow/launcher/AnalysisLauncher.java

**Output Location**

All results are saved in the exports/ directory. Outputs include:

|  |  |
| --- | --- |
| **File** | **Description** |
| failure\_cdf.csv | Time-to-failure distribution for the system or selected failure events |
| birnbaum.csv | Birnbaum importance measures for internal faults |
| fussell\_vesely.csv | Fussell-Vesely importance measures |
| cut\_sets.txt | Minimal cut sets (if configured) |
| execution\_time.txt | Execution time statistics |

**Summary**

This pipeline is organized into three stages:

**Stage 1 – Maintenance and Diagnostic Integration (Python)**  
Processes SAP maintenance records, diagnostic logs, and train compositions to calculate the time (in hours) between each maintenance event and the first occurrence of specific alarms.  
Output: Excel files per train, stored in the results/ folder.

**Stage 2 – Alarm Rate Estimation (Python)**  
Estimates the alarm occurrence rates (λ) based on Stage 1 results. Optional preprocessing fills missing values using the time to the next maintenance.  
Output: CSV file with λ values for each alarm (e.g., calculate\_statisctiche\_\*.csv).

**Stage 3 – Fault Modeling with FaultFlow (Java)**  
Uses the λ values in a fault model of the train's TCU system. The model is analyzed using FaultFlow to obtain failure distributions and fault importance measures.  
Output: Results saved in the exports/ folder.

This pipeline provides a systematic method to correlate train maintenance records with diagnostic alerts, taking into account the train's physical composition. It enables maintenance analysts and engineers to evaluate how quickly faults reappear after a repair, coach by coach, and supports further reliability analysis and preventive maintenance planning.