**🏆 TestAutothon 2025**

Conference Theme: AI-Powered Testing: Orchestrating Automation, Agents & Generative AI for the Future

Problem type: Non-AI problem

**⏱️ Timebox**

You have 3 hours total to:  
1. Understand the data and requirements  
2. Build your approach  
3. Prepare a short demo/presentation

**🎯 Challenge**

Build a complete solution that:

Turns a list of failing test incidents into a **deterministic, time-boxed test plan**. You will read a provided policy file (policy.yaml) and a dataset of failures, and then compute

1. How many minutes to allocate to additional testing for each incident and
2. A priority score to sort the work.

**Important:** You must base your calculations **only** on these incident fields:

* module
* environment
* failure\_type
* impacted\_layers

Ignore timestamps, logs, and any other fields.

No GenAI is needed for this problem.

1. Generate a dashboard that gives the complete information
2. Bonus: Integrate with Jira, Slack, Email, One click automation

**You will be given**

* **policy.yaml**: contains information based on which you need to create your solution
* **failures**.**jsonl**: 50+ incidents. Each JSON object includes at least the four required attributes listed above (plus other fields you must ignore).

**What you must build**

A small command-line script (single file is fine) that:

1. **Loads** policy.yaml and the failures file.
2. For **each incident**, computes:
   * **Minutes to Allocate for Additional Testing in an Iteration**:
     1. Base minutes are calculated as a sum of minutes for each layer in impacted layers.
     2. Final minutes are calculated by multiplying the above total minutes by environment multiplier and failure\_type multiplier.
     3. If the Final minutes are more than the upper cap mentioned in the policy file, then the upper cap is the one considered.
   * **Priority Score**:
     1. Calculated by retrieving the module’s priority and then multiplying it with environment multiplier and failure\_type multiplier
     2. If no module is mentioned then in the above calculation the module priority is considered as one before the multiplications involved.
3. **Outputs** a single machine-readable plan file (e.g., plan.json) containing an array of incident results with at least:
   * test\_id (if available)
   * module
   * environment
   * failure\_type
   * impacted\_layers
   * Base\_minutes **(for all the incidents)**
   * Final\_minutes **(for all the incidents)**
   * priority\_score (rounded to 3 decimals)
4. **Sorts** incidents by:
   * primary: priority\_score descending
   * secondary: module ascending (or any fixed, documented tie-breaker)

Keep the implementation simple and deterministic. No network calls, no external services.

**Constraints & rules**

* **Only** use: module, environment, failure\_type, impacted\_layers.
* **Ignore**: timestamps, or any other fields.
* **It should not be a GenAI** based solution.
* **Determinism**: the same inputs must always produce the same outputs.
* **Unknown keys**:
  + Unknown environment or failure\_type ⇒ multiplier = 1.0
  + Unknown module ⇒ module priority score = 1
  + No impacted\_layers entries means no testing is needed.

**Expected deliverables**

* **Source code** (single file is fine) with clear inline comments.
* **README.md** with:
  + How to run (commands, file paths).
  + Output format description.
  + Any assumptions you made (keep them minimal).
* **plan.json** (or .csv) produced from the provided dataset.
* (Optional) **Basic unit tests** validating a few representative cases.

**🏅 Competition Format**

- All teams compete in Round 1  
- Only the top teams advance to Round 2  
- Round 2 – better your solution, present to Jury  
**📊 Judging Criteria**

1. Completeness – End-to-end solution  
2. Accuracy – Correctness of calculations  
3. Innovation – Automation use  
4. Usability – Clarity of workflow/interface  
5. Presentation – Clear communication

**💡 Final Notes**

Be bold, creative, and practical.  
Done is better than perfect – aim for completeness first.