SituationReport

testdata_generator

User Manual

Generate synthetic Jira issue data for SituationReport

---

situation-report -- github.com/Jaegerfeld/situation-report

For Agile Coaches and PI Managers -- Version 0.19.0

1 What is the Testdata Generator?

The Testdata Generator creates synthetic Jira issue data in Jira REST API format. The generated files can be processed directly by transform_data — no real Jira instance required.

The generator simulates realistic workflow histories: issues traverse the defined stages, spend varying amounts of time in each stage, occasionally flow backwards, and close at a configurable rate.

> Typical use cases:
> - Test a new installation: Verify that transform_data and build_reports work correctly
> - Demos and presentations: Realistic data without privacy concerns
> - Training: Controlled learning environment with known data
> - Development: Reproducible test data via the seed parameter

The Testdata Generator complements the Get Data module (still in development), which loads real data directly from Jira. For the manual export of real Jira data, see the Get Data User Manual.

2 Prerequisites and Start

2.1 Portable package (recommended)

The portable package includes Python already bundled — no separate installation required.

Operating system	Start file
Windows	TestdataGenerator.bat (double-click)
macOS	TestdataGenerator.command (right-click → Open)
Linux	./TestdataGenerator.sh

Note for Windows: If double-clicking shows an error about Python not being found, the ZIP file may have been blocked by Windows. Fix: right-click the ZIP → Properties → Security → Unblock → OK → extract again.

2.2 From source code (Python required)

```
python -m testdata_generator
```

2.3 The user interface

After launch, the main window shows input fields for all parameters. Only the workflow file is required — all other fields have sensible defaults.

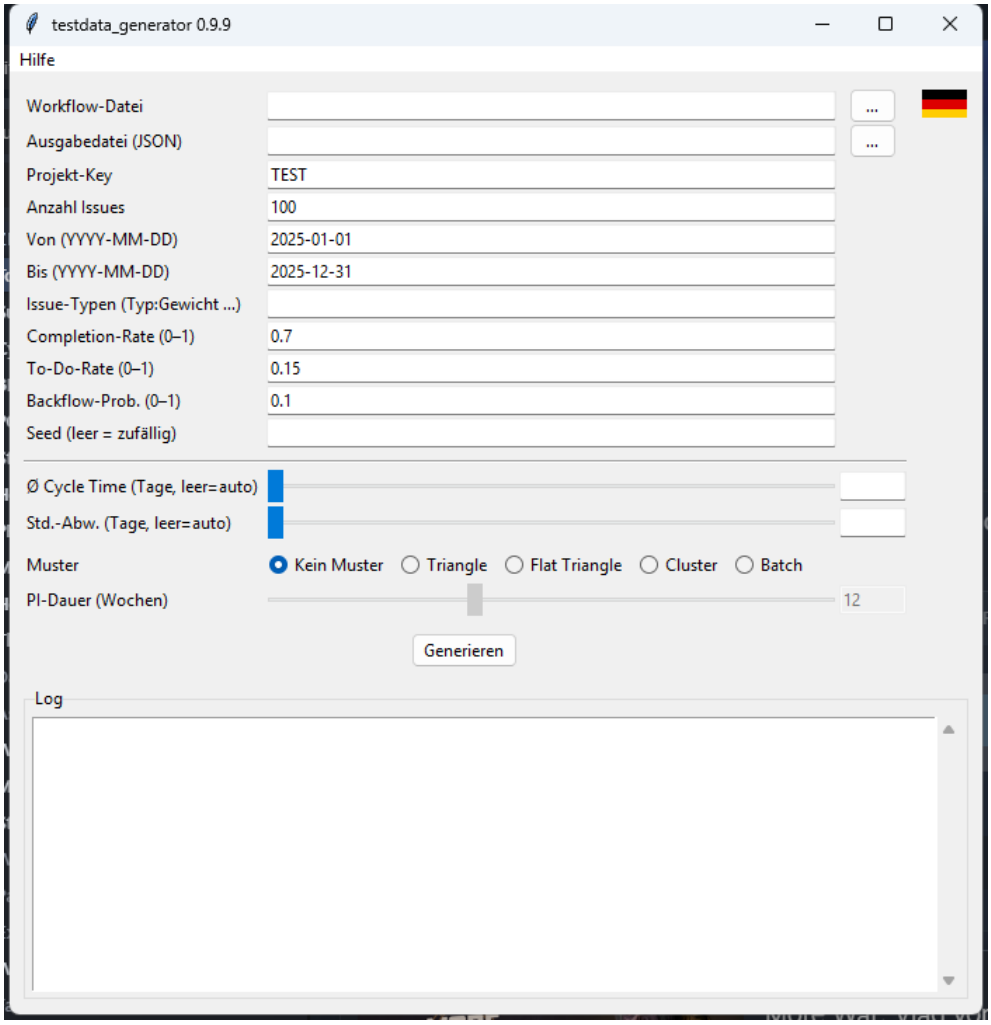

Fig. 1: Testdata Generator user interface

3 The Workflow File

The workflow file defines the stages of your Jira board and is the only required parameter. It uses the same format as transform_data — one file works for both modules.

3.1 Format

Each line describes a stage. Aliases (alternative status names in the Jira export) are separated by colons. Two special lines mark the start and end of active work:

```
CanonicalName:Alias1:Alias2
<First>FirstActiveStage
<Closed>ClosedStage
```

Line	Meaning
CanonicalName:Alias1:Alias2	Stage with one or more alternative Jira status names
StageName (no :)	Stage without aliases
StageName	First active stage — where processing begins (cycle time starts here)
StageName	Closed stage — marks an issue as completed

3.2 ART_A example workflow

The bundled example `workflow_ART_A.txt` models a typical SAFe ART workflow:

```
Funnel:New:Open:To Do
Analysis:In Analysis:Estimated
Program Backlog:Ready for Dev:Backlog
Implementation:In Implementation:In Review:In Progress
Blocker
Validating on Staging:QA
Deploying to Production:Ready for Development
Releasing:Completed
Done:Canceled
<First>Analysis
<Closed>Releasing
```

Notes: 'Funnel' has four aliases — all four Jira status names are normalised to 'Funnel'. Analysis is the first active stage (cycle time begins here). Releasing is the closed stage.

4 Parameter Reference

All parameters are available both in the GUI and on the command line.

Parameter	Default	Description
--workflow FILE	(required)	Workflow definition file (.txt)
--output FILE.json	_generated.json	Output file path
--project KEY	TEST	Jira project key for generated issues
--issues N	100	Number of issues to generate
--from-date YYYY-MM-DD	2025-01-01	Earliest creation date
--to-date YYYY-MM-DD	2025-12-31	Latest transition date
--issue-types TYPE:W ...	Feature:0.6 Bug:0.3 Enabler:0.1	Issue types with weights (sum need not equal 1)
--completion-rate FLOAT	0.7	Fraction of issues reaching the closed stage (0–1)
--todo-rate FLOAT	0.15	Fraction of open issues staying in early stages (0–1)
--backflow-prob FLOAT	0.1	Probability of a backward transition at each step (0–1)
--seed INT	(random)	Seed for reproducible output (optional)
--mean-cycle-days FLOAT	(none)	Target mean cycle time in days (lognormal); enables CT control
--std-cycle-days FLOAT	(30 % of mean)	Standard deviation of cycle time in days
--pattern PATTERN	none	Flow anti-pattern: none / triangle / flat_triangle / cluster / batch
--pi-duration-weeks INT	12	PI cycle length in weeks (for cluster/batch)
--pattern-strength 0-100	50	Pattern intensity 0–100 (0=subtle, 50=default, 100=strong)

Tip: Reproducible results
With a fixed seed (e.g. `--seed 42`), the generator always produces the same JSON file for the same parameters — ideal for demos where all participants should see identical results.

4b Flow Anti-Pattern Modes

Since version 0.9.9 the Testdata Generator can simulate typical flow anti-patterns from Vacanti and Singh. Combine --mean-cycle-days with --pattern. The GUI provides radio buttons and sliders for all pattern settings.

Pattern	Scatter plot appearance	Description
none	Even point cloud	Random cycle time without trend — default behaviour
triangle	Triangle: right angle bottom right	Cycle time rises linearly over time (multiplier ~1–4× at default strength, higher with --pattern-strength). Indicates a process that is becoming progressively slower.
flat_triangle	Triangle, rise flattens at the end	Like triangle but the increase is damped by tanh(2.5×t). Shows a process deterioration that is levelling off.
cluster	Vertical clusters of points at PI end	Cycle time stays stable (lognormal), but most issues are delivered in the last 2 weeks before the PI end. Deadline-driven behaviour.
batch	Columns of variable height at PI end	Like cluster but with highly variable cycle time (0.1× to 3× mean). Shows that both short and very long issues are pushed to the PI end together.

Cycle time control

When --mean-cycle-days is provided, cycle times are sampled from a lognormal distribution with the given mean.

Parameter	Meaning
--mean-cycle-days	Mean cycle time in days (e.g. 20). Required for all patterns except none.
--std-cycle-days	Standard deviation in days (default: 30 % of mean). Higher values produce a wider spread in the scatter plot.
--pi-duration-weeks	Length of one PI cycle in weeks (default 12). Only relevant for cluster and batch.

Examples

```
# Triangle pattern: CT rises from 20d to 80d
python -m testdata_generator \
    --workflow workflow.txt --issues 300 \
    --pattern triangle \
    --mean-cycle-days 20 --std-cycle-days 6 \
    --completion-rate 0.8 --seed 1 \
    --output triangle.json

# Cluster of Dots: deliveries at PI end (12-week PI)
python -m testdata_generator \
    --workflow workflow.txt --issues 300 \
    --pattern cluster \
    --mean-cycle-days 30 --pi-duration-weeks 12 \
    --completion-rate 0.8 --seed 2 \
    --output cluster.json

# Batch Transfers: PI end + high CT variance
python -m testdata_generator \
    --workflow workflow.txt --issues 300 \
    --pattern batch \
    --mean-cycle-days 30 --pi-duration-weeks 12 \
    --completion-rate 0.8 --seed 3 \
    --output batch.json
```

Note: --pattern requires --mean-cycle-days. Without --mean-cycle-days the pattern is ignored.

5 ART_A – Complete Example

This example shows the full pipeline from generator to finished report using the fictional ART_A project.

Step 1: Generate test data

```
python -m testdata_generator \
    --workflow testdata_generator/workflow_ART_A.txt \
    --project ART_A \
    --issues 200 \
    --from-date 2025-01-01 \
    --to-date 2025-12-31 \
    --issue-types Feature:0.6 Bug:0.2 Enabler:0.2 \
    --completion-rate 0.75 \
    --backflow-prob 0.08 \
    --seed 42 \
    --output ART_A_generated.json
```

Result: `ART_A_generated.json` with 200 issues — approximately 150 closed and 50 open. Reproducible with seed 42.

Step 2: Process with transform_data

```
python -m transform_data ART_A_generated.json \
    --workflow testdata_generator/workflow_ART_A.txt
```

Step 3: Create report with build_reports

```
python -m build_reports
```

> Expected output (seed 42, 200 issues):
> - Issue keys: ART_A-0001 to ART_A-0200
> - Issue types: ~120 Feature, ~40 Bug, ~40 Enabler
> - ~150 issues with status 'Releasing' or 'Done'
> - Creation dates distributed across 2025
> - Occasional backward transitions (~8 %)

5b The Portfolio Scenario

With `--scenario portfolio` the generator creates a complete demo portfolio in one step: two solutions with three ARTs each, including every artifact of the processing chain.

In the GUI: the 'Demo portfolio' section at the bottom of the window generates the scenario into a folder of your choice at the click of a button (the seed field is honoured, default 42); 'Open Portfolio Report' renders the portfolio report and opens it in the browser — no command line needed.

```
python -m testdata_generator --scenario portfolio \
    --output demo/ --seed 42
```

The target folder then contains:

Artifact	Content
workflows/	two workflow files (Alpha and Beta stages)
raw/	six Jira JSON exports (one generator run per ART)
arts/	IssueTimes, CFD, and Transitions files per ART
solution_alpha.json	solution config without stage_map (default path)
solution_beta.json	solution config with its own stage_map (schema 2)
risks_*.json	ROAM risk register per solution
nfr_*.json	NFR/runway register per solution
capabilities_*.json	capability map per solution
dependencies_*.json	dependency register per solution
decisions_*.json	decision/assumption log per solution
portfolio.json	portfolio config covering both solutions
pi_config.json	PI intervals across the data window
README.md	description of the built-in stories

The data is placed relative to the generation date so the portfolio report's quality traffic light rates it as current. Three stories are built in:

Built-in stories:
- ART Alpha-3 is the outlier (about three times the cycle time) — highlighted red in Solution Alpha's comparison report.
- ART Beta-3 delivers weak data (no CFD, few started issues, data 60 days old) — confidence 'low' in the quality table.
- Solution Beta pools via its own stage_map (Vorlauf/Umsetzung/Fertig); Solution Alpha uses the default path.
- ROAM board: two owned risks are deliberately old (45/50 days) — the aging highlight fires.
- NFR & runway: Beta's API NFR is violated and one runway element is an overdue gap — the dashboard shows red.
- Capability map: Beta's data-insights capability is critical, one Alpha capability has no ART (uncovered).
- Dependency heatmap: Alpha-1 → Alpha-3 blocked and overdue; Beta-1 → Alpha-1 as a cross-solution integration.
- Decision log: Beta's open assumption has passed its review date — flagged red as 'review due'.

The folder is directly usable: `python -m portfolio demo/portfolio.json` builds the portfolio report; the solution configs also work individually. The same seed yields identical data on the same day.

6 Frequently Asked Questions (FAQ)

Difference between test data and real data?

For demos, training, and installation tests, test data is the better choice — no privacy concerns, reproducible, and fully controlled. For real flow analyses and decisions, actual data reflects the true workflow state.

For the manual export of real Jira data: See the Get Data User Manual.

7 Glossary

Term	Explanation
ART	Agile Release Train — a team-of-teams structure in SAFe
API	Application Programming Interface — programming interface
CFD	Cumulative Flow Diagram — cumulative issue count per stage over time
Cycle time	Time from the first active stage to the closed stage
Helper	SituationReport module for merging multiple Jira JSON files
Issue	A work item in Jira (Feature, Bug, Enabler, Story, …)
JSON	JavaScript Object Notation — format of Jira API exports
Seed	Starting value for the random generator — same seed = same output
Stage	One step in the workflow (corresponds to a Jira column)
Workflow	Ordered sequence of stages that an issue passes through