AmiEs 2026

A Trade-off-Aware Approach for Prioritized Refactoring of Dockerfiles

Delcio Cosme Monjane · Isabel Azevedo

ISEP — Instituto Superior de Engenharia do Porto

AmIEs 2026

github.com/delcioMonjane/dmei2

Overview

1. **Why It Matters** — Slides 1–4
   The problem: smells are detected but never ranked or explained.
2. **The Approach** — Slides 5–9
   A trade-off-aware prioritization model on top of existing detectors.
3. **The Tool & Results** — Slides 10–13
   Docker Prioritizer in action, evaluated on 57 real Dockerfiles.
4. **Closing** — Slides 14–16
   Limitations, contributions, and what comes next.

What is a Dockerfile smell?

```
FROM python
RUN apt-get update && apt-get install -y curl
ADD requirements.txt /app/requirements.txt
RUN pip install -r /app/requirements.txt
COPY . /app
WORKDIR /app
USER root
CMD ["python", "app.py"]
```

- ! No version pinned on the base image — resolves to :latest
- ! Missing --no-install-recommends and apt cache cleanup
- ! ADD used where COPY would suffice
- ! No non-root user ever created

These are Dockerfile smells — violations of recommended practice that quietly degrade security, performance, maintainability, or reproducibility. Empirical studies report that most open-source Dockerfiles contain at least one.

The problem

Same Dockerfile, run through Hadolint:

Dockerfile:1	**DL3007**	warning	Using latest is prone to errors, pin the version
Dockerfile:3	**DL3015**	info	Avoid additional packages with --no-install-recommends
Dockerfile:3	**DL3008**	warning	Pin versions in apt-get install
Dockerfile:3	**DL3009**	info	Delete the apt lists after installing
Dockerfile:5	**DL3020**	**error**	Use COPY instead of ADD for files and folders
Dockerfile:6	**DL3042**	warning	Avoid cache directory with pip: use --no-cache-dir
Dockerfile:11	**DL3002**	warning	Last USER should not be root

Seven findings. No ranking, no severity that reflects real impact, no indication that fixing one could hurt another. Which one do you start with?

The trade-off

Repair action

Pin package versions

Reproducibility +5

Same input, same image, every time.

Maintainability −1

Dependencies stop updating on their own.

Security −2

No more automatic upstream patches.

One action, three directions. A tool that only reports "Reproducibility improved" hides the maintenance and security cost being paid for it — and vice-versa.

Goal and research questions

Design, implement, and evaluate a prioritization mechanism that ranks Dockerfile repairs by their expected impact on four quality attributes — making the trade-offs between them explicit, instead of hiding them inside a single score.

RQ1 Which Dockerfile smells have the most significant impact on key quality attributes such as performance, maintainability, reproducibility and security?

RQ2 How can a prioritization mechanism be designed for Dockerfile refactoring that balances trade-offs between quality attributes and incorporates developer's contextual preferences?

RQ3 What is the effectiveness of the prioritized, trade-off-aware approach in improving Dockerfile quality, and what evidence exists for its practical applicability?

What already exists

Tool	Detection	Automated repair	Prioritization	Trade-offs
Hadolint	✓	–	–	–
Binnacle	✓	–	–	–
Parfum	✓	✓	–	–
DockerCleaner	✓	✓	–	–

None of the four rank smells or surface trade-offs between quality attributes.

Approach overview



One pipeline: an external detector feeds a literature-derived, empirically calibrated impact matrix; developer-defined weights turn it into one ranked list with every trade-off visible.

The impact matrix

Repair action	Security	Performance	Maintainability	Reproducibility
Add --no-install-recommends	0	+4	0	0
Remove package manager caches	0	+5	+1	0
Replace ADD with COPY	0	−1	+4	+2
Run as non-root user	+5	0	+1	0
Remove explicit platform constraints	0	0	+3	+2
Pin dependency versions	−2	0	−1	+5

Derived from the systematic literature review, on a -5..+5 scale. Empirical calibration (§4.4.3) tested these values against real repairs and did not find strong enough evidence to revise them.

How priority is calculated

$$\text{Priority}(s) = \sum_{q} wq \times I(s, q)$$

wq developer-defined weight for quality attribute q **I(s, q)** impact of smell s on attribute q, from the calibrated matrix

A Weighted Sum Model — chosen over AHP and TOPSIS for its transparency: no pairwise comparison matrices, no extra normalization step, and every attribute's contribution stays inspectable in the final score.

The tool at work

`poetry run docker-prioritizer prioritize Dockerfile.example — the same Dockerfile shown earlier`

`Detected 5 smells in Dockerfile.example`

1 **DL3002** Switching to the root USER opens up certain security risks if an attacker gets access to the container. **−10.00**

`Security −10`

2 **aptGetInstallUseNoRec** Using --no-install-recommends with apt-get install saves layer space and reduces attack surface. **−10.00**

`Security −3 · Performance −7`

3 **aptGetInstallThenRemoveAptLists** Running rm -rf /var/lib/apt/lists/* after apt-get install improves efficiency and image size. **−9.00**

`Performance −9`

⋮ #5 pipUseNoCacheDir (−8.00) omitted here — no trade-off, same pattern as #1–#3

4 **DL3020** Fix for 'DL3020' on line 5: Use COPY instead of ADD for files and folders. **−8.00**

`Security −6 Performance +0 Maintainability +2 Reproducibility −4`

Using ADD instead of COPY is more convenient (auto-fetch URLs, auto-extract archives), but that same flexibility widens the attack surface and harms reproducibility — fetched content can differ between builds in ways a plain COPY never would.

Real output, real Dockerfile — 5 smells detected, ranked, scored, and the ADD/COPY trade-off spelled out in full.

How it was evaluated

1

Clone & build

57 Dockerfiles from open-source repos, baseline build

→

2

Detect & repair

Parfum detects smells, logs predicted impact, applies repair

→

3

Rebuild

3 cold builds per Dockerfile; cache pruned before each

→

4

Compare

Median build time / image size retained, before vs. after

EQ1 Does automated refactoring reduce image size, build time, and layer count, and improve abstract quality scores?

EQ2 Are the observed effects consistent, in direction and magnitude, with the calibrated impact matrix?

Results: the effect of repair (EQ1)

−115 s

average build-time reduction

in the 24 of 57 Dockerfiles where a smell was actually repaired

−123 MB

average image-size reduction

same 24-of-57 repaired subset

24 of 57 Dockerfiles had at least one smell actually repaired.

16 of 57 had no detected smell (no-op). 17 of 57 did not complete the before/after cycle (build regressions or pre-existing failures).

Results: predicted vs. observed (EQ2)

Held-out sample of 11 Dockerfiles, not used to calibrate the impact matrix

Quality attribute	Direction match	% match	Correlation (r)	
Security	11/11	100%	**1.00**	
Performance	10/11	91%	**0.73**	
Maintainability	9/11	82%	**0.63**	
Reproducibility	9/11	82%	**undefined**	*least reliable*

Reproducibility is the weakest signal: predicted impact and observed change disagreed in 2 of 11 cases, and near-zero variance in this sample leaves the correlation undefined. Stated plainly, not hidden.

Limitations

1

Small held-out sample

EQ2 validation rests on n = 11 Dockerfiles not used for calibration.

2

Reproducibility stayed static

Layer count did not change in any of the 40 repaired Dockerfiles, and neither did the Reproducibility score.

3

Single point of failure

The whole pipeline depends on Parfum as its detection oracle — a known FROM-instruction defect there caused most non-completions.

Conclusion

1. A calibrated, trade-off-aware impact matrix combining literature evidence with empirical validation.
2. A Weighted-Sum-Model prioritization mechanism that keeps every attribute's contribution, and every trade-off, inspectable.
3. Docker Prioritizer: an end-to-end CLI tool, evaluated on 57 real-world Dockerfiles.

Repaired Dockerfiles have been submitted as pull requests — work in progress, not yet a result: maintainer responses have not been received.

github.com/delcioMonjane/dmei2

Thank you

Questions?

Delcio Cosme Monjane
1211739@isep.ipp.pt

Isabel Azevedo
ifp@isep.ipp.pt

`github.com/delcioMonjane/dmei2`

Appendix

Supporting data

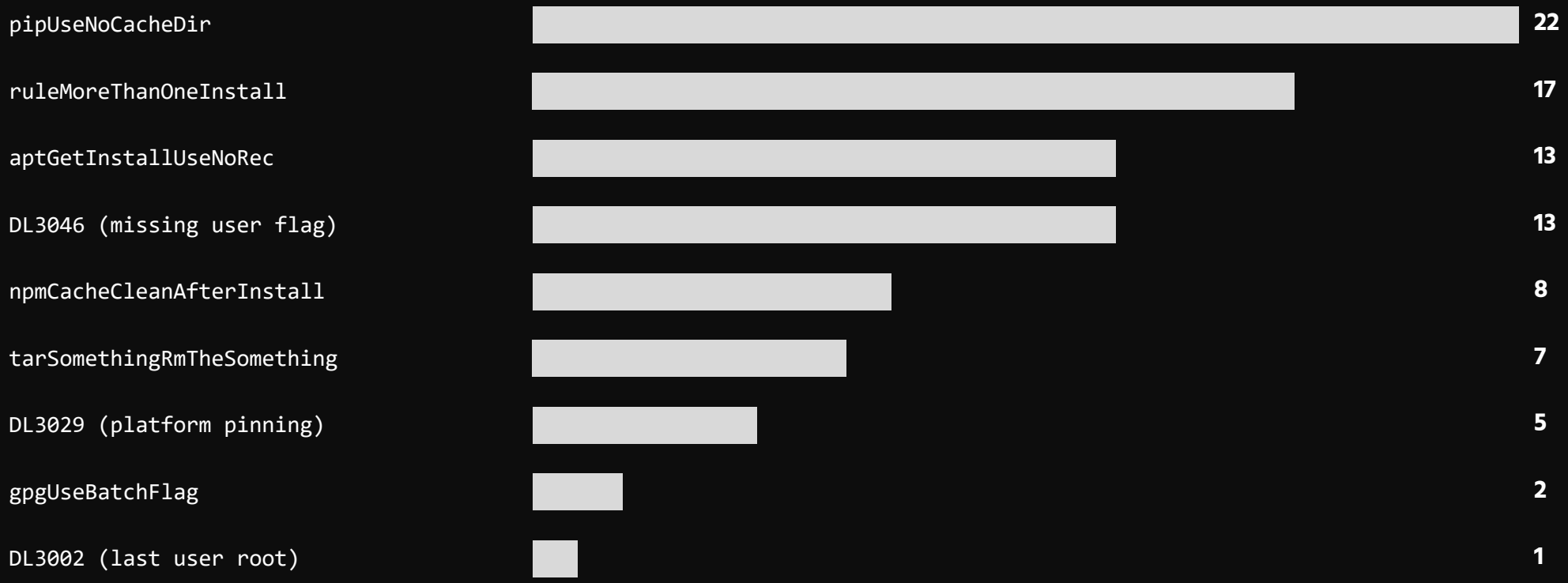
Full impact matrix (Table 7)

Repair action	Security	Performance	Maintainability	Reproducibility
Merge package installation commands	0	+4	+2	0
Add apt-get update before install	0	0	0	+4
Add --no-install-recommends	0	+4	0	0
Remove package manager caches	0	+5	+1	0
Replace HTTP with HTTPS	+5	0	0	0
Replace ADD with COPY	0	−1	+4	+2
Remove sudo usage	+4	0	+2	0
Run as non-root user	+5	0	+1	0
Remove explicit platform constraints	0	0	+3	+2
Pin dependency versions	−2	0	−1	+5

Table 7, Chapter 4. Identical to the initial literature-derived matrix (Table 4): calibration found no strong enough evidence to revise any value at the current sample size.

Smell type distribution across the corpus

88 instances, 9 smell types, across 36 Dockerfiles with at least one logged smell



The top 3 types account for over 60% of instances — all Performance-related, matching Performance's largest average score gain.

Why 17 of 57 Dockerfiles did not complete

40	7	10
Completed the full cycle	**SYNTAX_REGRESSION**	**PARSE_ERROR**
built, repaired, rebuilt, re-measured	built before repair, broke after Parfum's repair	original Dockerfile never built, independent of repair

Root cause of SYNTAX_REGRESSION: in the earlier 28-Dockerfile batch, 4 of 5 regressions shared an identical Docker parser error on the FROM instruction — pointing to a systematic defect in how Parfum rewrites FROM lines. The full 57-Dockerfile run logs only the failure category, not the raw error text, so this specific pattern could not be re-confirmed at this scale; retaining the raw error message is recommended future work.

Detailed tools comparison (Table 3)

Tool	Strengths	Limitations	License & maintenance
Hadolint	Widely adopted; CI/CD integration; evolving rules	Treats all violations equally; no repair	GPL-3.0, actively maintained
Binnacle	Expands catalogue beyond Hadolint via expert-mined rules	No repair; limited adoption; dataset-dependent	Research artifact, no license, inactive since 2020
Parfum	Automated repair; reduces image size	~7% build failures; efficiency smells only	Public repo, no license, research artifact
DockerCleaner	Targets vulnerabilities; improves security posture	Security smells only; potential build instability	Public repo, no license, ICSME 2023 artifact

Systematic review methodology

PRISMA-guided selection, August 2025 – April 2026, Google Scholar + IEEE Xplore + ACM Digital Library

7,590	records identified
420	after dedup / accessibility filtering
95	after title & abstract screening
20	primary studies included

Framework: Kitchenham & Charters' SLR guidelines, within a Design Science Research methodology (6 phases: problem identification, objectives, design, demonstration, evaluation, communication).

Live-demo backup: actual console output

```
PS C:\Users\delci\Documents\ISEP\MEI\2oAno\dmei2> poetry run docker-prioritizer prioritize Dockerfile.example
[dim]Running command: C:\Program Files\nodejs\docker-parfum.CMD analyze C:\Users\delci\Documents\ISEP\MEI\2oAno\dmei2\Dockerfile.example[/dim]
Detected 5 smells in Dockerfile.example
```

Priority	Smell & Repair	Score	Impact	Trade-offs
1	**DL3002** Fix for 'DL3002' on line 11: Switching to the root USER opens up certain security risks if an attacker gets access to the container. In order to mitigate this, switch back to a non privileged user after running the commands you need as root.	-10.00	Security: -10 | Performance: +0 | Maintainability: +0 | Reproducibility: +0	None
2	**aptGetInstallUseNoRec** Fix for 'aptGetInstallUseNoRec' on line 3: Using the --no-install-recommends flag with apt-get install in a Dockerfile helps save layer space, improve build times, and reduce the size and attack surface of the final image, as well as prevent hidden dependencies.	-10.00	Security: -3 | Performance: -7 | Maintainability: +0 | Reproducibility: +0	None
3	**aptGetInstallThenRemoveAptLists** Fix for 'aptGetInstallThenRemoveAptLists' on line 3: Running rm -rf /var/lib/apt/lists/* after apt-get install in a Dockerfile can improve efficiency and reduce the size of the image.	-9.00	Security: +0 | Performance: -9 | Maintainability: +0 | Reproducibility: +0	None
4	**DL3020** Fix for 'DL3020' on line 5: Use COPY instead of ADD for files and folders.	-8.00	Security: -6 | Performance: +0 | Maintainability: +2 | Reproducibility: -4	Using ADD instead of COPY is more convenient (it can fetch URLs and auto-extract archives), but that same flexibility widens the attack surface, harming security. ADD's auto-extraction and remote-fetch convenience comes at the cost of reproducibility: the extracted contents or fetched resource can differ between builds in ways a plain COPY of a local file never would.
5	**pipUseNoCacheDir** Fix for 'pipUseNoCacheDir' on line 6: Using the --no-cache-dir flag with pip in a Dockerfile can disable the package cache, ensuring that the latest version of a package and its dependencies are installed.	-8.00	Security: +0 | Performance: -8 | Maintainability: +0 | Reproducibility: +0	None

```
Analysis saved with ID: dac78ee2-7767-43a2-a730-927c4c8d6bb6
```

poetry run docker-prioritizer prioritize Dockerfile.example — unedited terminal capture, kept as a fallback if the live demo (slide 10) cannot run in the room.