Fair Developer Score: Build-Adjusted Effort & Impact

Rethinking Developer Productivity Metrics in the AI Era

ASE 2025 – ISE Workshop

Xinzhou Wang · Jiancong Zhu · Jinghan Feng · Zixuan Zhang · Joshua Rauvola · Devon Delgado · Ahmad Antar · Abid Ali

AI tools went from niche to default in just three years

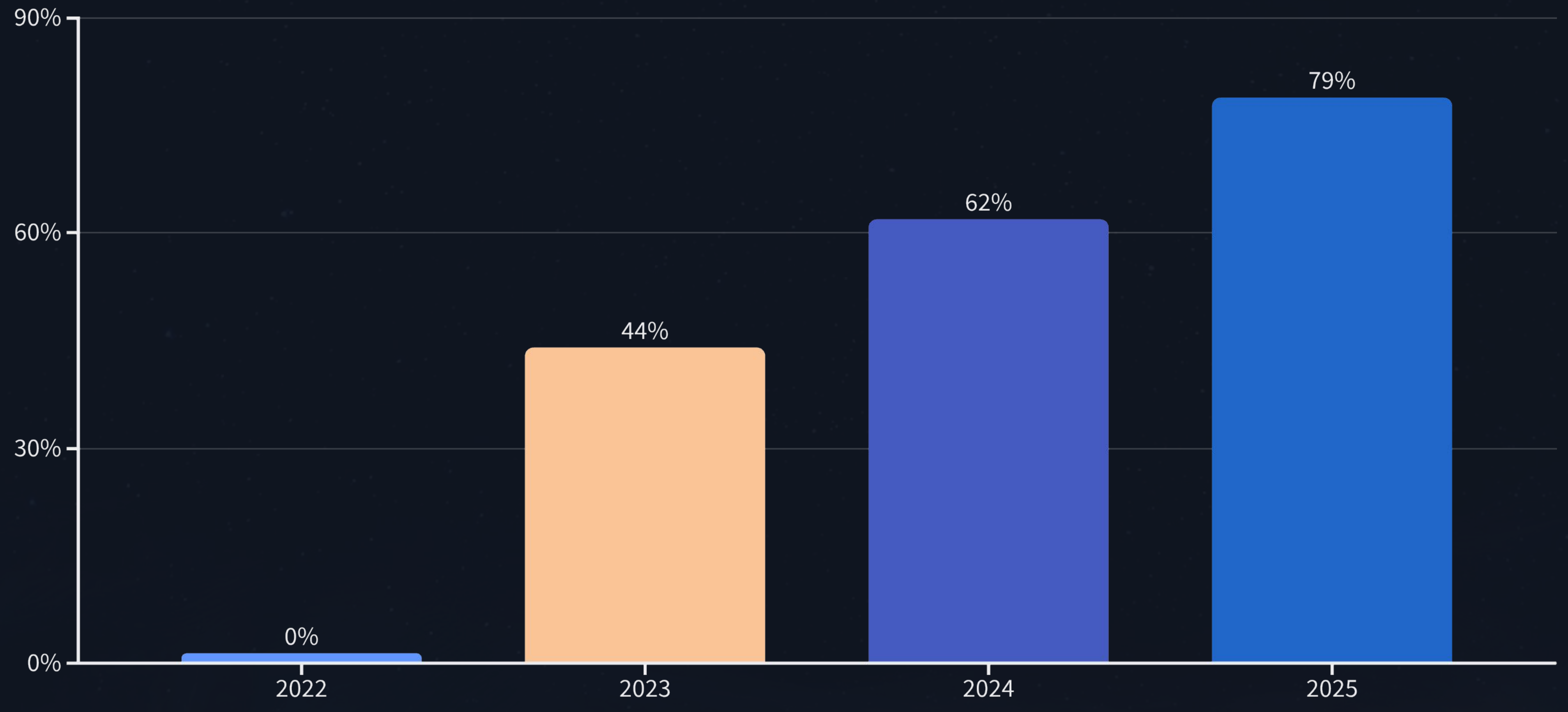
Share of developers currently using AI tools – Stack Overflow Developer Surveys 2022–2025

Summary

Stack Overflow data shows rapid adoption: 44% → 62% → ~79% of developers are now using AI tools (2023–2025).

Impact

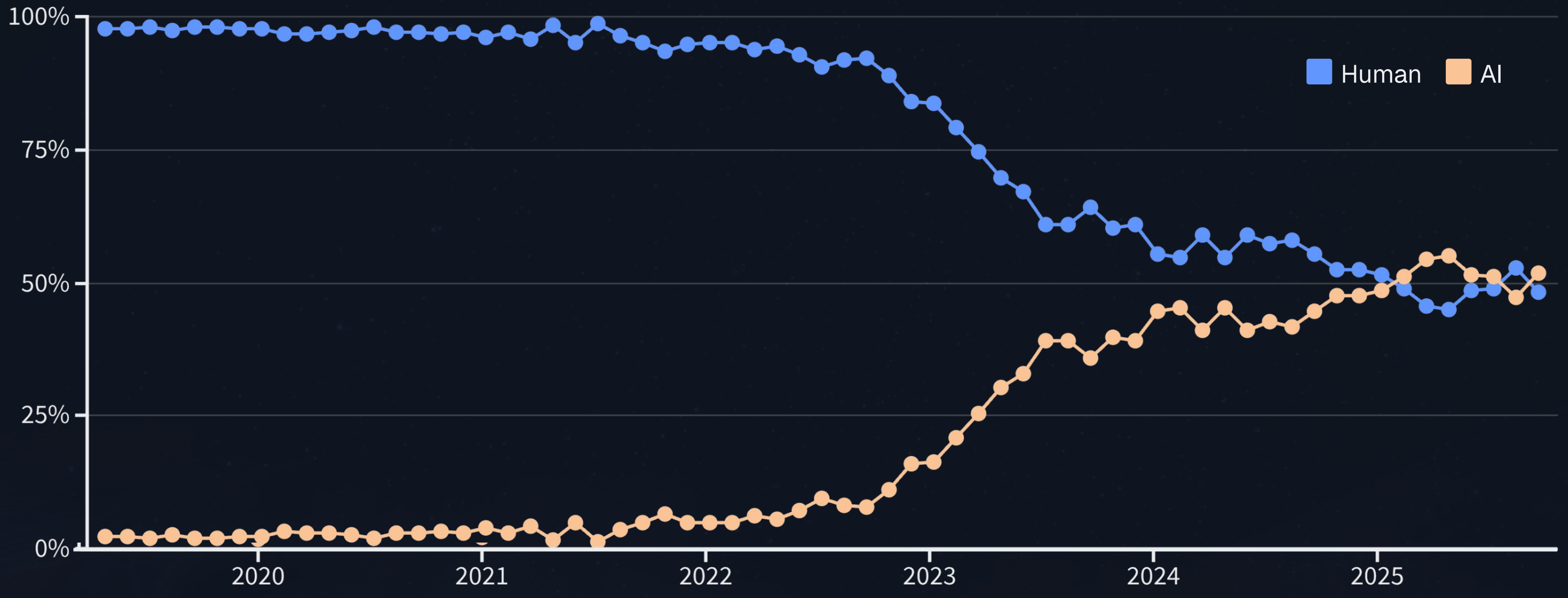
AI support moved from experimental to default in the developer workflow in just three survey cycles.



Annotations: 2022: Pre-mainstream baseline (≈0%, not directly measured) 2023: 44% of developers already using AI tools 2025: ~79% of developers using AI tools at least sometimes

AI-generated content has rapidly caught up with human content

Monthly share of AI vs human-written articles (Graphite.io, 2020–May 2025)



This chart vividly illustrates the rapid rise of AI in content generation. While AI contributed less than 5% of new articles in early 2020, its share dramatically increased, especially after the launch of ChatGPT in late 2022. By late 2024, AI-generated articles began to surpass human-written ones, marking a significant shift in the content landscape.

The Evolving Software Development Landscape

79%

AI Tool Usage

Developers now use AI coding assistants daily

40%

AI-Generated Code

New code comes from using AI under heavy usage in large companies such as Coinbase

30%

Bot Contributions

A large share of PRs now come from bots, inflating activity metrics.

180M

Platform Scale

Active developers across GitHub alone

Traditional individual metrics struggle to remain meaningful in this new environment where human and AI contributions intertwine across massive collaborative ecosystems.

Introducing the Fair Developer Score

The Fair Developer Score (FDS) is a **commit-centric, build-adjusted framework** that fundamentally rethinks productivity measurement by combining two orthogonal dimensions.

Developer Effort

How much and how substantively each developer contributed to a build

Build Importance

How impactful that build is to the system and organization

FDS aggregates these dimensions by summing **Effort × Importance** across all builds. This approach is lightweight and generalizable – it uses only commit metadata, requiring no raw source code or proprietary ticket systems, making it broadly deployable across open-source and enterprise environments.

The FDS Pipeline: From Commits to Score

Commit Clustering

Group related commits into logical builds using Torque Clustering

Effort Scoring

Compute each developer's effort per build using standardized multi-factor analysis

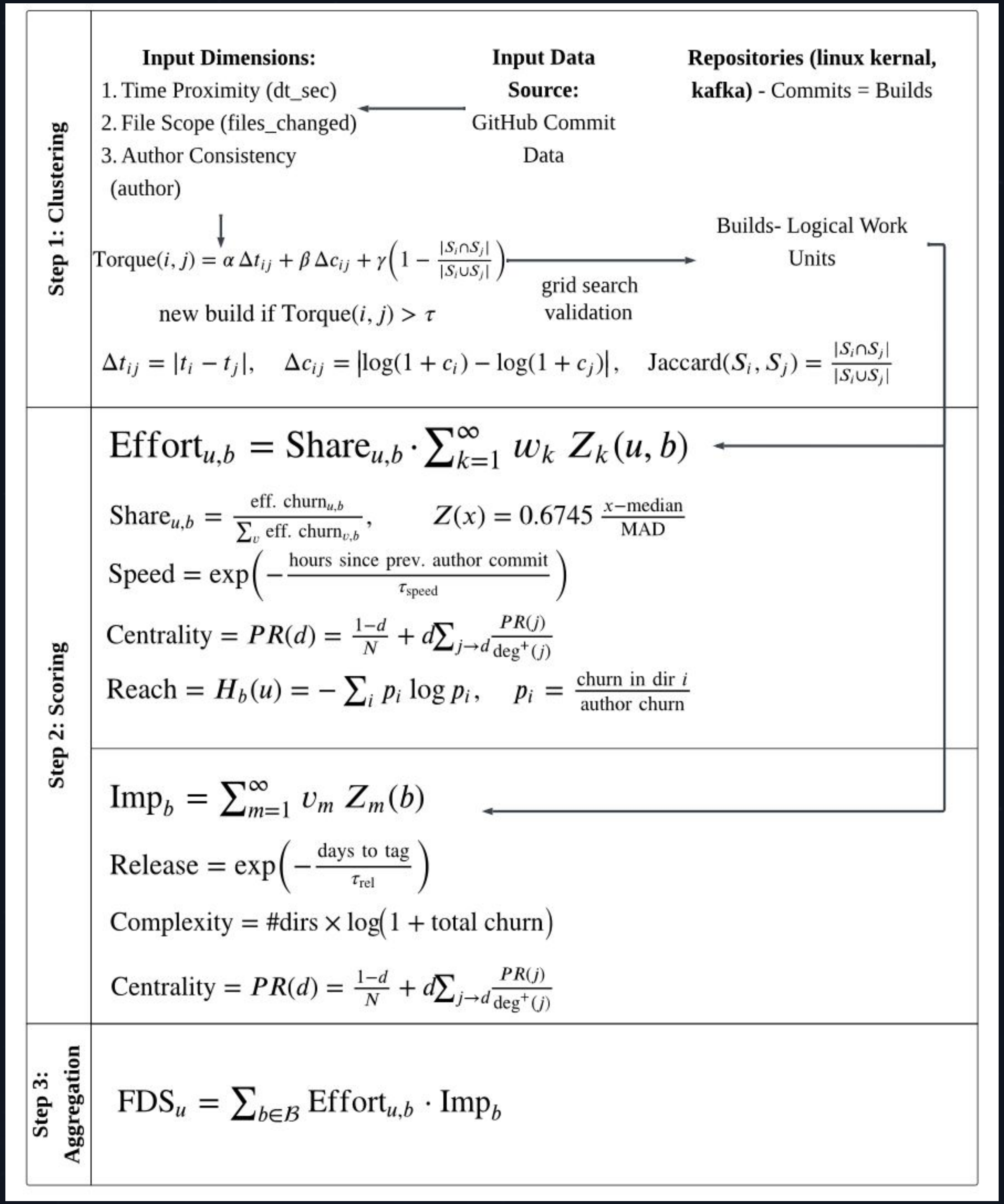
Importance Scoring

Assess each build's impact through structural and semantic features

Aggregation

Sum Effort × Importance across all builds to produce FDS

This pipeline transforms raw version control history into meaningful, context-aware productivity measurements that capture both the quantity and quality of developer contributions.



Commit Clustering: Defining Logical Work Units

A **build** represents a logical work unit – a feature, bug fix, or refactor – that may span multiple commits. We use Torque Clustering to identify these natural boundaries in the commit stream.

The torque between consecutive commits *i* and *j* combines temporal and change magnitude signals:

$$Torque(i,j) = \alpha \Delta t_{i,j} + \beta \Delta c_{i,j}$$

When *Torque(i,j) > gap*, we start a new build. Large jumps in time or change magnitude signal task boundaries.

Our implementation also considers file scope shifts and author continuity as heuristics, capturing multi-commit features as single cohesive builds for more realistic task-level analysis.

Developer Effort: Multi-Dimensional Contribution

Effort captures each developer's contribution to a build beyond simple line counts, combining six standardized dimensions weighted to reflect true productivity:

$$Effort_{u,b} = Share_{u,b}(w_1 Z_{Scale} + w_2 Z_{Reach} + w_3 Z_{Centrality} + w_4 Z_{Dominance} + w_5 Z_{Novelty} + w_6 Z_{Speed})$$

Scale

Size of changes measured through logarithm of churn

Reach

Breadth of impact across directory and file hierarchy

Centrality

Importance of edited components via PageRank on co-change graph

Leadership

Leadership in build initiation and finalization

Novelty

Introduction of new modules and APIs

Speed

Rapid, focused commit cadence indicating concentrated work

All features use MAD–Z standardization for robustness against heavy-tailed distributions, preventing any single metric from dominating the score.

Build Importance: Quantifying Task Impact

While Effort is developer-specific, **Importance** is a property of the build itself, measuring how significant a given task is to the project:

$$Importance_b = 0.30Z_{Scale} + 0.20Z_{Scope} + 0.15Z_{Centrality} + 0.15Z_{Complexity} + 0.10Z_{Type} + 0.10Z_{Release}$$

Scale (30%)

Total churn in the build reflecting overall size

Scope (20%)

How widespread the change across files and subsystems

Centrality (15%)

Effect on architecturally core components

Complexity (15%)

Joint signal of size multiplied by dispersion

Type (10%)

Priority classification from commit messages

Release (10%)

Temporal proximity to major release milestones

The weights reflect a research-validated prior: scale and scope matter most, while architectural position, complexity, priority, and timing provide essential context.

Fair Developer Score: The Effort–Impact Fusion

The Fair Developer Score elegantly combines individual effort with task impact through multiplication and summation across all builds:

$$FDS_u = \sum_b Effort_{u,b} \times Importance_b$$

What This Achieves

- Rewards high-effort work on high-importance builds
- De-emphasizes low-value churn and peripheral contributions
- Balances quantity with quality and context
- Moves from activity measurement to outcome alignment

Key Insight: The same effort on low-importance work contributes less to FDS, and low-effort involvement in important builds contributes less as well. This multiplicative relationship addresses fundamental biases in traditional volume-based metrics.

Validation: Linux Kernel Case Study

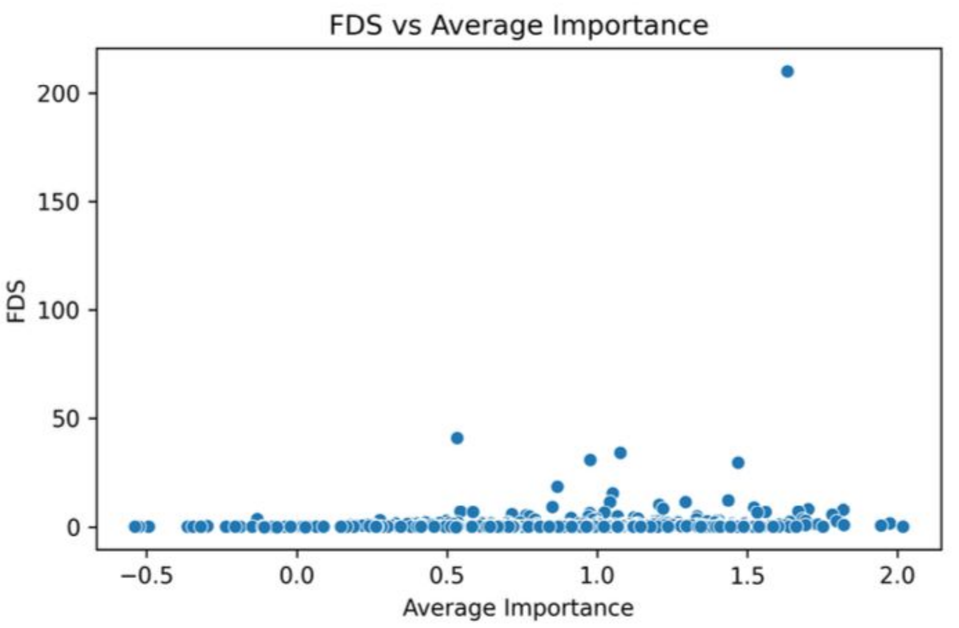


Fig. 4. FDS vs Average Importance

TABLE III
MATCHED TOP-DECILE: **AVERAGE EFFORT** (MAD–Z). Δ IS FDS–COMMIT.

Repo	n	Δ	p	Cliff's δ	95% CI (Δ)
Linux Kernel	34	0.079	0.002	0.32	[0.039, 0.125]
Kubernetes	20	0.058	0.012	0.40	[0.023, 0.095]
TensorFlow	15	0.058	0.028	0.40	[0.011, 0.115]
Apache Kafka	23	0.055	0.028	0.26	[0.016, 0.105]
PostgreSQL	16	0.034	0.109	0.19	[0.000, 0.072]

We validated FDS on the Linux kernel – one of the largest and longest-running open-source projects – analyzing 974 days of development across 339 contributors.

Rigorous Matched-Pair Design

To ensure fair comparison, we matched top-decile FDS developers one-to-one with top-decile commit-count developers using the Hungarian algorithm, controlling for total churn, files changed, and unique builds participated in.

Higher Average Importance

FDS-ranked developers work on systematically more impactful builds at equal volume

Higher Average Effort

More substantive contributions per build, reflecting deeper engagement

Lower Rework Rate

Fewer short-interval directory revisits, indicating more sustainable contributions

Cross-Repository Evaluation

To test generalizability, we applied the same methodology across four additional major open-source projects: Kubernetes, TensorFlow, Apache Kafka, and PostgreSQL.

TABLE I
REPOSITORY CHARACTERISTICS FOR CROSS-VALIDATION

Repository	Lang	N	Days	FDS Range	Med
Linux Kernel	C	339	974	0.011–210.2	0.35
Kubernetes	Go	199	1,321	0.013–333.9	0.30
TensorFlow	Py/C++	146	746	0.012–453.2	0.27
Apache Kafka	Java/Scala	225	589	0.010–33.30	0.35
PostgreSQL	C	31	562	0.057–159.1	5.89

OS Kernel, Container orchestration, ML framework, Dist. systems, RDBMS

1 Linux Kernel
- ✓ Effort significant
- ✓ Importance significant

2 Kubernetes
- ✓ Effort significant
- ✓ Importance significant

3 Apache Kafka
- ✓ Effort significant
- ✓ Importance significant

4 TensorFlow
- ✓ Effort significant
- ○ Importance mixed

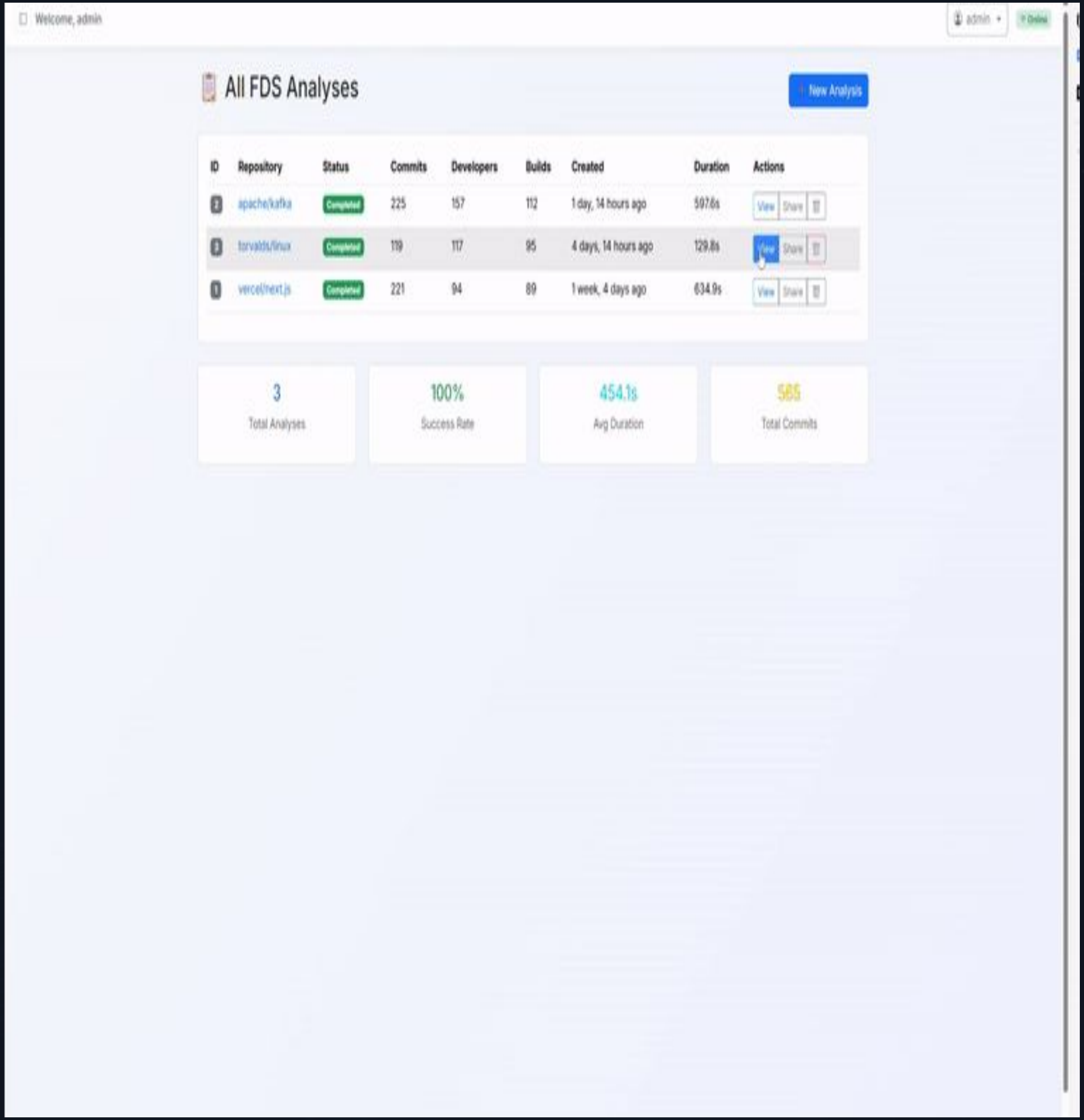
5 PostgreSQL
- ✓ Effort significant
- ○ Small sample

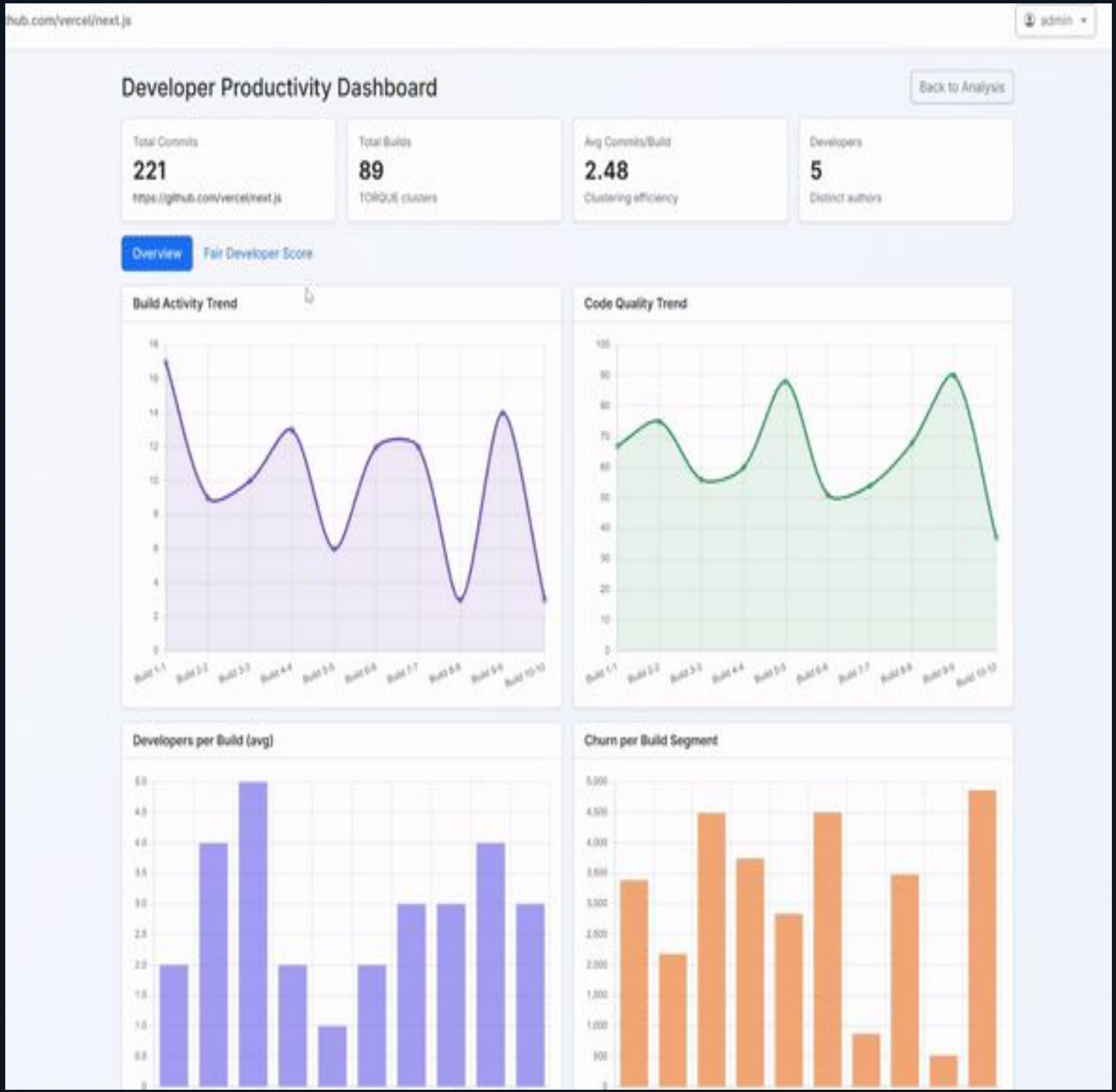
Key Findings

The **Effort model generalizes robustly** – in all five repositories, top FDS developers show higher Effort than volume-matched commit-count peers, with statistical significance in four of five projects.

The **Importance signal is context-sensitive**, performing best in Linux, Kubernetes, and Kafka. TensorFlow's heavy automation and generated code patterns require Importance model tuning.

This pattern validates our design: Effort is universal, while Importance should adapt to project-specific workflows and automation profiles.


Welcome, admin
admin
All FDS Analyses
New Analysis
ID
Repository
Status
Commits
Developers
Builds
Created
Duration
Actions
apache/kafka
Completed
225
157
112
1 day, 14 hours ago
597.6s
View
Share
torvalds/linux
Completed
119
117
95
4 days, 14 hours ago
129.8s
vercel/next.js
Completed
221
94
89
1 week, 4 days ago
634.9s
3
Total Analyses
100%
Success Rate
454.1s
Avg Duration
565
Total Commits


hub.com/vercel/next.js
admin
Developer Productivity Dashboard
Back to Analysis
Total Commits
221
https://github.com/vercel/next.js
Total Builds
89
TORQUE clusters
Avg Commits/Build
2.48
Clustering efficiency
Developers
5
Distinct authors
Overview
Fair Developer Score
Build Activity Trend
Code Quality Trend
Developers per Build (avg)
Churn per Build Segment

Conclusion & Future Outlook

What FDS Delivers Today

- Build-adjusted, commit-only metric combining Developer Effort and Build Importance
- Validated differentiation of core from peripheral contributors across five major repositories
- Practical framework requiring only version control metadata

Known Limitations

- Commit-only view misses qualitative factors like mentoring and design work
- Some parameters are expert-chosen and benefit from project-specific tuning

The AI Era Imperative

As AI reshapes software development, our assessment frameworks must evolve. Future directions include:

1. Distinguishing human versus AI-generated code in Effort calculations
2. Tracking technical debt and quality implications of AI assistance
3. Integrating with DevEx and SPACE frameworks for satisfaction and cognitive load

---

FDS offers a practical step toward fairer, impact-aligned measurement – moving beyond naive activity counts to recognize meaningful contributions in an increasingly complex development landscape.

Appendix

TABLE III

MATCHED TOP-DECILE: **AVERAGE EFFORT** (MAD–Z). Δ IS FDS–COMMIT.

Repo	n	Δ	p	Cliff's δ	95% CI (Δ)
Linux Kernel	34	0.079	0.002	0.32	[0.039, 0.125]
Kubernetes	20	0.058	0.012	0.40	[0.023, 0.095]
TensorFlow	15	0.058	0.028	0.40	[0.011, 0.115]
Apache Kafka	23	0.055	0.028	0.26	[0.016, 0.105]
PostgreSQL	16	0.034	0.109	0.19	[0.000, 0.072]

TABLE IV

REWORK RATIO (PP): REVISIT SAME DIR WITHIN 48H. LOWER IS BETTER.

Repo	n	Δ pp	p	Cliff's δ	95% CI (Δ)
Linux Kernel	34	−4.5	0.069	−0.15	[−8.9, −0.1]
Kubernetes	20	−5.6	0.263	−0.10	[−13.8, 2.3]
TensorFlow	15	1.9	0.249	0.13	[−0.9, 5.1]
Apache Kafka	23	1.6	0.917	−0.09	[−2.9, 7.1]
PostgreSQL	16	1.8	0.285	0.06	[0.0, 4.8]

TABLE V

ROLLBACK RATE (PP): COMMITS WITH 'REVERT' IN SUBJECT. LOWER IS BETTER.

Repo	n	Δ pp	p	Cliff's δ	95% CI (Δ)
Linux Kernel	34	−0.4	0.180	−0.06	[−1.1, 0.0]
Kubernetes	20	0.7	0.180	0.10	[0.0, 1.9]
TensorFlow	15	0.4	0.317	0.07	[0.0, 1.2]
Apache Kafka	23	0.0	0.655	0.00	[−0.3, 0.3]
PostgreSQL	16	0.4	0.109	0.19	[0.0, 1.0]

TABLE VI

EFFECT SIZES FOR META-ANALYSIS (CLIFF'S δ).

Repo	n	δ_{Imp}	δ_{Eff}
Linux Kernel	34	0.26	0.32
Kubernetes	20	0.40	0.40
TensorFlow	15	−0.27	0.40
Apache Kafka	23	0.17	0.26
PostgreSQL	16	0.06	0.19