DEPSLY RESEARCH SERIES

Research Brief 01

Dependency governance benchmark

State of Dependency Governance 2026

Findings from 100 JavaScript and TypeScript repositories

100	83%	41%	94%
Repositories analyzed	Crossed the deep-graph threshold	Showed a concentration hotspot	Produced nontrivial review work

EXECUTIVE ABSTRACT

This report examines structural dependency complexity across a benchmark cohort of 100 JavaScript and TypeScript repositories. The central finding is not simply that vulnerabilities matter. It is that governance becomes difficult when deep graphs, transitive reliance, and concentrated package influence become the norm.

Prepared from a private benchmark dataset generated with the local Depsly analysis engine.

WHAT THIS REPORT COVERS

- Large dependency graphs are normal, not exceptional, in modern JavaScript and TypeScript projects.
- Transitive complexity dominates the median repository, which weakens direct-only review models.
- A small set of high-influence packages repeatedly shapes graph exposure across unrelated projects.

BENCHMARK COHORT: 100 REPOSITORIES

FOCUS: DEPENDENCY GOVERNANCE, TRANSITIVE COMPLEXITY, AND REVIEWABILITY

STATUS: DRAFT RESEARCH BRIEF

Key Findings

SCALE

The median repository in the cohort contained 1,570.5 dependency nodes, with a long tail of substantially larger graphs.

TRANSITIVE RELIANCE

The median transitive ratio was 0.7071, which means inherited complexity dominates many real projects.

DEPTH

Eighty-three percent of repositories crossed the current deep-graph threshold, making traceability and review harder.

CONCENTRATION

Forty-one percent of repositories showed a concentration hotspot where one package influenced at least 10 percent of the graph.

Why this report exists

Open source dependency review is often framed as a vulnerability problem. That framing is too narrow for organizations that need to explain decisions, retain evidence, and revisit dependency posture over time.

This benchmark is intended to give compliance teams, security practitioners, and engineering leaders a common baseline for discussing dependency governance as an operating problem, not just a scanning problem.

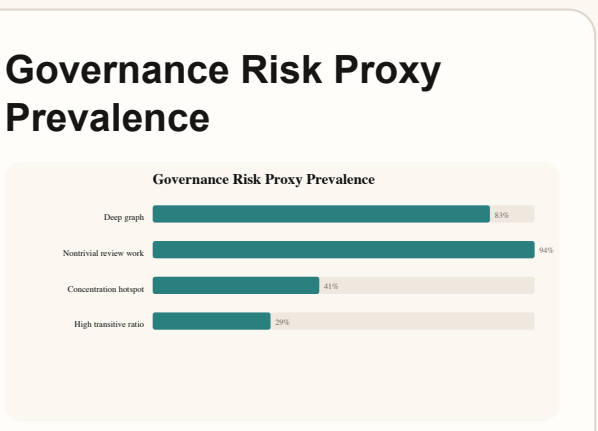
Who should read it

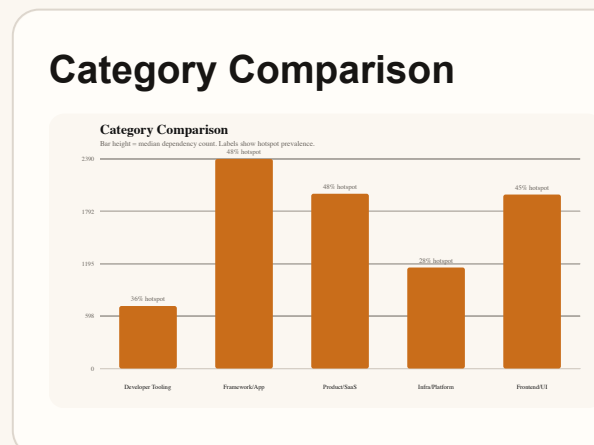
- Compliance and GRC teams that need durable review evidence
- Security and AppSec teams that need structural risk context beyond CVEs
- Engineering leaders responsible for dependency debt and maintenance posture

Contents

- Executive Summary
- Why Dependency Governance Deserves Its Own Lens
- Methodology
  - Scope
  - Analysis Engine
  - Metrics Used
  - Important Limitations
- Benchmark Findings
  - 1. Large dependency graphs are normal
  - 2. Transitive complexity dominates real projects
  - 3. Deep graphs are widespread
  - 4. Concentration hotspots are common
  - 5. Review work is almost always present
  - 6. A few transitive packages repeatedly shape graph exposure
- Category Comparisons
  - Framework / App Surface
  - Developer Tooling
  - Infra / Platform
  - Product / SaaS-Style Applications
  - Frontend / UI Ecosystem
- What These Findings Mean For Governance
  - CVEs are necessary but insufficient
  - Dependency oversight is often a workflow problem
  - Governance maturity should include structure, not only findings
- Recommendations By Audience
  - For Compliance / GRC Teams
  - For Security / AppSec Teams
  - For Engineering Leadership
- Conclusion

At a Glance


Governance Risk Proxy Prevalence


Category Comparison

State of Dependency Governance 2026

Executive Summary

Open source dependency review is usually framed as a vulnerability problem.

That framing is incomplete.

Across a benchmark cohort of `100` JavaScript and TypeScript repositories, the dominant pattern was not simply the presence or absence of CVEs. It was structural complexity:

- large dependency graphs
- deep transitive chains
- repeated concentration around a small set of highly influential packages
- widespread need for review work that is difficult to manage without a durable workflow

In the benchmark dataset:

- the median repository contained `1570.5` dependency nodes
- the median repository contained `765.5` transitive dependencies
- the median transitive ratio was `0.7071`
- the median maximum dependency depth was `17`
- `83%` of repositories crossed the current deep-graph threshold
- `41%` showed a concentration hotspot where one package affected at least 10% of the dependency graph
- `94%` produced at least one nontrivial recommendation requiring action beyond simple deferral

These results suggest that many organizations do not just have a vulnerability management problem. They have a dependency governance problem: how to review, explain, prioritize, document, and revisit decisions in environments where transitive complexity is the norm.

Why Dependency Governance Deserves Its Own Lens

Security and compliance teams often inherit dependency oversight through adjacent programs:

- software composition analysis
- vulnerability review
- procurement or vendor review
- engineering risk management
- audit evidence collection

But dependency governance is not reducible to any one of those functions.

A repository can be free of headline vulnerabilities and still be difficult to govern if:

- the graph is unusually deep
- most of the footprint is transitive
- a few packages exert broad downstream influence
- review decisions are not retained as evidence over time

That is why dependency governance deserves its own lens. The question is not only whether a package is vulnerable. The question is whether the organization can understand, review, and govern the dependency structure it relies on.

Methodology

Scope

This benchmark reflects `100` repositories that:

- validated successfully under current lockfile support
- produced complete machine-readable scan outputs
- span framework, tooling, infra, product, and frontend ecosystem categories

Category counts in the final cohort:

- `framework_app`: `23`
- `developer_tooling`: `25`
- `infra_platform`: `18`
- `product_app`: `23`
- `frontend_ui_ecosystem`: `11`

Analysis Engine

All scans were produced with the `depsly` CLI using committed project lockfiles as the source of dependency structure.

Metrics Used

The benchmark focused on structural and governance-adjacent metrics, including:

- total dependency count
- transitive dependency count
- transitive ratio
- maximum dependency depth
- blast-radius concentration
- recommendation prevalence

Important Limitations

This report is not a claim about the entire JavaScript ecosystem.

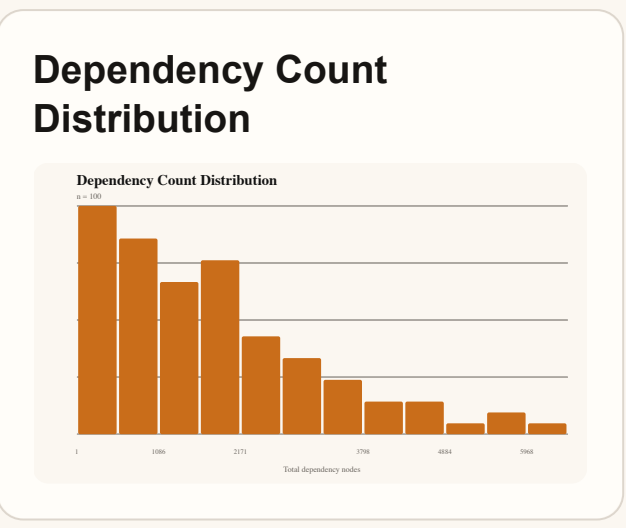
The final cohort is shaped by:

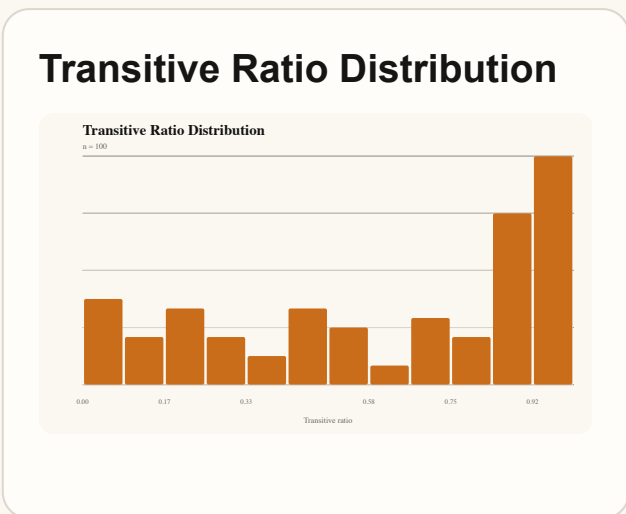
- repository availability
- committed lockfile format
- parser support boundaries
- scanability under current tooling

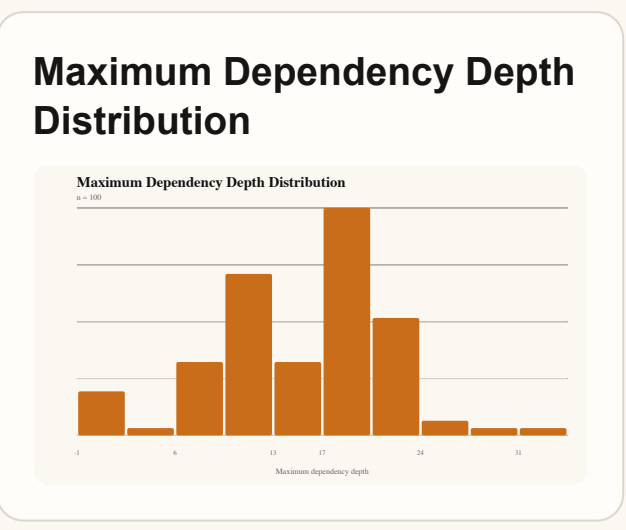
The pipeline currently performs best on:

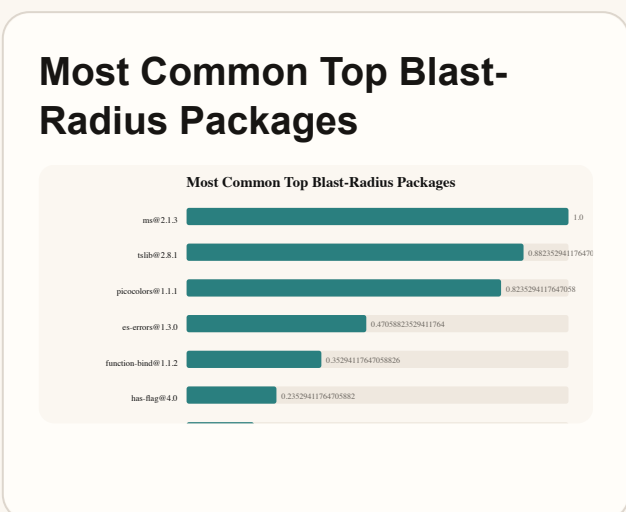
- `pnpm-lock.yaml`
- modern `package-lock.json`
- a subset of `yarn.lock` repositories compatible with current parser support

That means the sample is research-grade but not ecosystem-exhaustive. The results are best interpreted as a benchmark of real, scanable projects under current support boundaries rather than a universal census.


Dependency Count Distribution


Transitive Ratio Distribution


Maximum Dependency Depth Distribution


Most Common Top Blast-Radius Packages

Benchmark Findings

1. Large dependency graphs are normal

The median repository contained `1570.5` dependency nodes. The mean was `1789.44`, indicating a long tail of especially large graphs.

Percentile view:

- p25: `716.75`
- p50: `1570.5`
- p75: `2473.75`

The main governance implication is straightforward: many teams inherit graphs that are already large before any vulnerability-specific investigation begins.

2. Transitive complexity dominates real projects

The median repository contained `765.5` transitive dependencies, and the median transitive ratio was `0.7071`.

Percentile view:

- p25 transitive ratio: `0.3262`
- p50 transitive ratio: `0.7071`
- p75 transitive ratio: `0.9086`

This matters because most of the operational footprint in many projects is not directly chosen. It is inherited through layered dependency chains.

That weakens simplistic control assumptions such as:

- only reviewing direct dependencies
- treating top-level manifests as a complete picture
- assuming intake approval is enough

3. Deep graphs are widespread

Using a working threshold of `max_depth >= 10`, `83%` of repositories in the dataset qualified as deep graphs.

Depth distribution:

- p25: `12`
- median: `17`
- p75: `20`

Deep graphs matter because they increase the difficulty of:

- tracing why a package exists
- understanding downstream impact
- explaining review decisions
- stabilizing remediation work when upstream packages change

4. Concentration hotspots are common

`41%` of repositories showed a concentration hotspot, meaning a single package affected at least 10% of the dependency graph under the current threshold.

This is not just a security observation. It is a governance observation.

When a small number of packages carry disproportionate structural influence:

- review effort becomes more concentrated
- upstream changes can propagate more broadly
- a small number of packages may deserve disproportionate scrutiny

5. Review work is almost always present

`94%` of repositories produced at least one nontrivial recommendation.

This suggests that the real bottleneck is often not whether reviewable work exists. It is whether the organization has a repeatable system for:

- deciding what to review
- documenting the decision
- retaining the rationale
- comparing the current state with prior scans

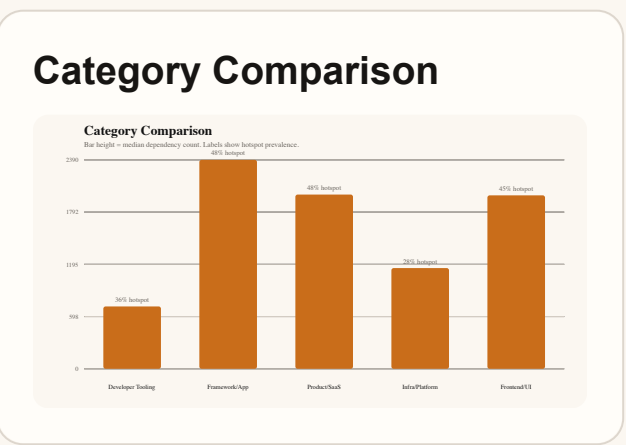
6. A few transitive packages repeatedly shape graph exposure

The most common top blast-radius packages in the benchmark were:

- `ms@2.1.3` (`17`)
- `tslib@2.8.1` (`15`)
- `picocolors@1.1.1` (`14`)
- `es-errors@1.3.0` (`8`)
- `function-bind@1.1.2` (`6`)

This does not make them a vulnerability ranking.

It does make them useful evidence that certain packages repeatedly occupy structurally influential positions across real-world dependency graphs.


Category Comparison

Category Comparisons

Dependency governance is not uniform across project types.

Framework / App Surface

- count: `23`
- median total dependencies: `2390`
- median transitive ratio: `0.4623`
- median max depth: `19`
- concentration hotspot prevalence: `47.8%`

Interpretation:

- framework-heavy repositories are often very large
- even when direct ownership is relatively high, depth and concentration remain meaningful governance issues

Developer Tooling

- count: `25`
- median total dependencies: `713`
- median transitive ratio: `0.8421`
- median max depth: `12`
- concentration hotspot prevalence: `36%`

Interpretation:

- tooling repos tend to be smaller in absolute node count
- but they often rely more heavily on transitive layers

Infra / Platform

- count: `18`
- median total dependencies: `1152`
- median transitive ratio: `0.8540`
- median max depth: `19`
- concentration hotspot prevalence: `27.8%`

Interpretation:

- platform and infra-adjacent repos show strong transitive reliance and deep graphs
- even without the highest hotspot prevalence, they remain structurally difficult to review

Product / SaaS-Style Applications

- count: `23`
- median total dependencies: `1993`
- median transitive ratio: `0.6667`
- median max depth: `16`
- concentration hotspot prevalence: `47.8%`

Interpretation:

- product applications combine high absolute dependency volume with meaningful concentration risk
- this is one of the most buyer-relevant categories for governance messaging

Frontend / UI Ecosystem

- count: `11`
- median total dependencies: `1984`
- median transitive ratio: `0.5377`
- median max depth: `19`
- concentration hotspot prevalence: `45.5%`

Interpretation:

- frontend ecosystem repos are often large and consistently deep
- they may look manageable at the top level while still carrying structurally complex graphs

What These Findings Mean For Governance

CVEs are necessary but insufficient

Vulnerability information remains important. But it is not enough to describe how governable a dependency graph actually is.

The benchmark suggests organizations also need to understand:

- graph depth
- transitive reliance
- concentration hotspots
- repeated high-influence packages

Dependency oversight is often a workflow problem

The widespread presence of nontrivial recommendations suggests that useful review work exists in many projects today. What is often missing is a durable process around it.

Mature dependency governance should be able to answer:

- What was reviewed?
- Why was a decision made?
- Who owned it?
- What changed since the last review?
- What evidence would support that answer in an audit or customer review?

Governance maturity should include structure, not only findings

Organizations often assess dependency health through issue counts alone. This benchmark suggests structural complexity is an important part of the picture.

Two projects can have the same number of known issues while differing materially in:

- reviewability
- traceability
- operational fragility
- concentration of transitive influence

Recommendations By Audience

For Compliance / GRC Teams

- treat dependency review as a recurring evidence-producing process, not a one-time control
- retain scan history, comparison context, reviewer notes, and follow-up state
- ask not only whether vulnerabilities exist, but whether reviewability and ownership are demonstrable

For Security / AppSec Teams

- add structural metrics to the review lens, especially depth, transitive ratio, and concentration hotspots
- prioritize packages that repeatedly occupy high-influence graph positions
- separate vulnerability severity from structural influence in internal communications

For Engineering Leadership

- treat dependency structure as a maintainability and operational risk issue, not just a security issue
- review high-depth and high-fanout areas as technical debt
- build workflows for repeated scan comparison rather than relying on isolated one-off reviews

Conclusion

The benchmark does not support the claim that dependency oversight can be reduced to CVE review alone.

Across 100 JavaScript and TypeScript repositories, the dominant pattern was structural complexity:

- large graphs
- deep graphs
- heavy transitive reliance
- common concentration hotspots
- widespread nontrivial review work

That is why dependency governance deserves to be treated as its own discipline, with its own evidence, workflows, and operational maturity model.

Appendix: Benchmark Tables

These tables summarize the cohort-level benchmark values used throughout the report. They are intended as reference material for readers who want the underlying distribution and prevalence figures in one place.

Benchmark Tables

Summary

Metric	Value
Repositories analyzed	100
Median total dependencies	1570.5
Median transitive dependencies	765.5
Median transitive ratio	0.7071
Median max depth	17.0
Concentration hotspot prevalence	41%
Deep graph prevalence	83%
Nontrivial recommendation prevalence	94%

Category Snapshot

Category	Count	Median Total Dependencies	Median Transitive Ratio	Median Max Depth	Hotspot Prevalence
Developer Tooling	25	713	0.8421	12	36%
Framework / App	23	2390	0.4623	19	48%
Product / SaaS	23	1993	0.6667	16	48%
Infra / Platform	18	1152	0.854	19	28%
Frontend / UI	11	1984	0.5377	19	45%

Local draft generated from the private 100-repository benchmark dataset.