
All about dependencies

Ixchel Ruiz · ixchel.ruiz@karakun.com



Ixchel Ruiz

- linkedin.com/in/ixchelruiz
- @ixchelruiz@mastodon.social
- github.com/ixchelruiz



Human!

**WRITE CODE FOR HUMANS
TO IMPROVE HUMAN LIFE**

Human!

IMPROVE HUMAN LIFE

PART I

Dependencies as a Condition of Software

Dependencies are architectural commitments.

They provide capability and leverage, today.

**The obligations arrive later: understanding, maintenance, compatibility, security, support,
communication, eventual replacement.**

Dependencies are borrowed architecture.

**The product incorporates decisions made by
other people, projects, companies, and communities.**

Why dependency is necessary

From individual payoff to economic leverage.

- 01 Speed: skip rebuilding what is already solved**
- 02 Tested functionality: already proven at scale**
- 03 Specialist capability: expertise we lack in house**
- 04 Interoperability: shared formats and protocols**
- 05 Economic leverage: cost shared across many adopters**

Reuse reduces the cost of construction,

while increasing the number of concerns the product must manage.

Working Knowledge

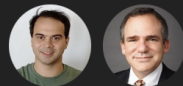
Distilling Harvard Business School research for leaders who make a difference

[Lead with Research](#) ▾ [Popular Research](#) [About Working Knowledge](#)

Entrepreneurship

Open Source Software: The \$9 Trillion Resource Companies Take for Granted

Many companies build their businesses on open source software, code that would cost firms \$8.8 trillion to create from scratch if it weren't freely available. Research by **Frank Nagle** and colleagues puts a value on an economic necessity that will require investment to meet demand.



Featuring Manuel Hoffmann and [Shane M. Greenstein](#). By Rachel Layne on March 22, 2024

When we add a dependency, are we adding *code*, or are we adding a *relationship*?



An author unpublished eleven lines of code, and thousands of builds broke within hours across the JavaScript ecosystem.

left-pad

npm, March 2016. Verify exact scope and affected projects before presenting.



What I regret,

Handing off a project to the support engineering team that included development-time dependencies

The fewer dependencies I include, the greater the flexibility I return to the team.

What the relationship contains

From day one to the whole horizon.

01 Codebase

02 Release process

03 License

04 Platform assumptions

05 Maintainers

06 Vulnerability disclosure process

07 Service availability

08 Lifecycle

“We use a library” becomes “we participate in a network of obligations.”



Complexity “The structure of a software system that makes it hard to understand and modify.

John Ousterhout.

A Philosophy of Software Design, 2018



PART II

The Dependency Relationship

Dependency manager: ***“A mechanism for improving the flexibility and comprehensibility of a system.”***

David L. Parnas

On the Criteria to Be Used in Decomposing Systems into Modules, CACM, 1972

Dependency as an organizational boundary

- 01** Maintainers
- 02** Vendors
- 03** Package registries
- 04** Cloud providers
- 05** Standards bodies

**A dependency is either a promise
we intend to keep,**

or a convenience nobody agreed to maintain.

**How much control are we handing
to people we have never met?**

Dependency and trust

01 Authorship

02 Release

03 Transport

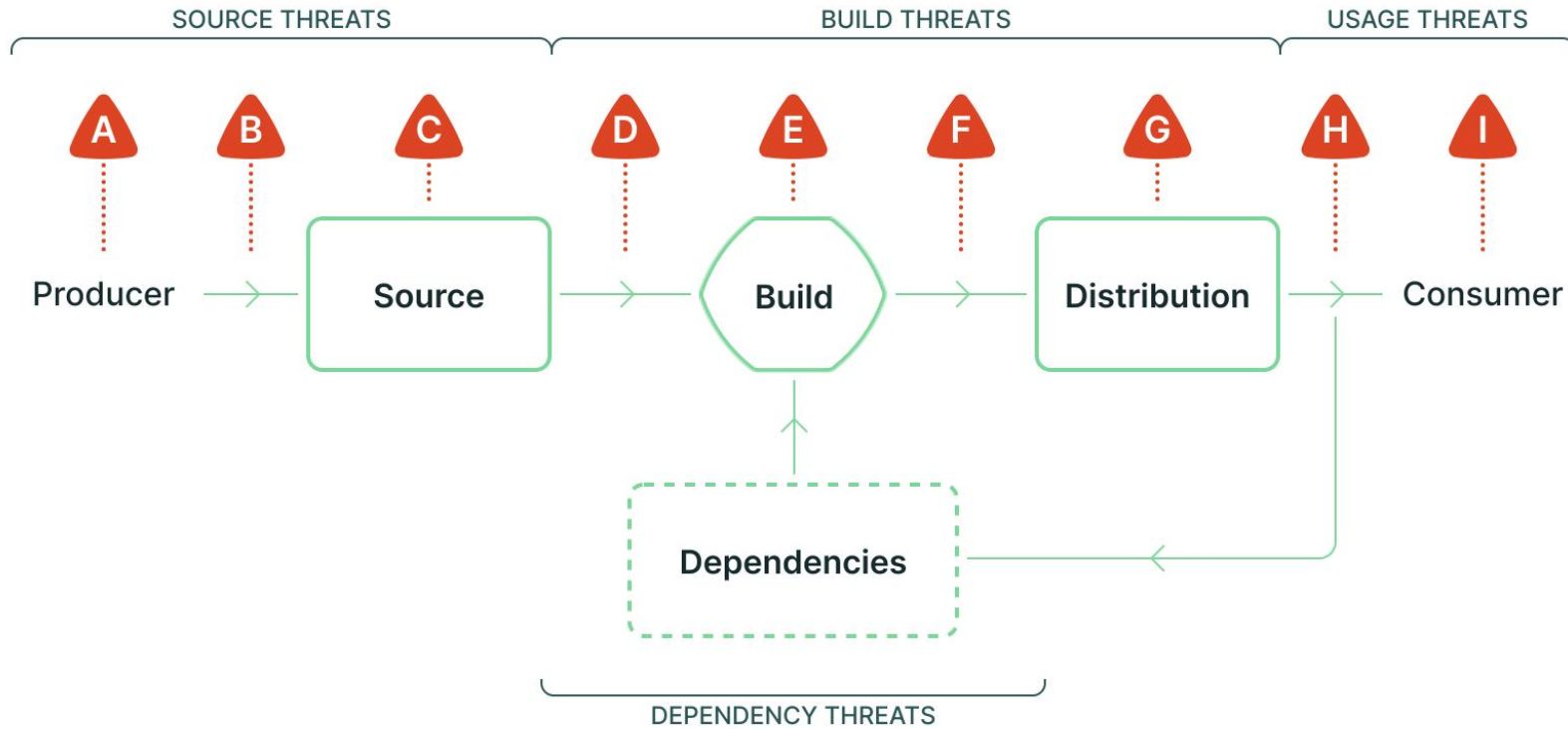
04 Resolution

05 Installation

06 Execution

07 Maintenance

The relationship is temporal. It changes as maintainers, versions, platforms, and expectations change.



A Producer (entity)
B Authoring & reviewing
C Source code management

D External build parameters
E Build process
F Artifact publication

G Distribution channel
H Package selection
I Usage



Two years of patient trust-building turned out to be a nation-state operation, caught only by a half-second delay in SSH logins.

xz-utils backdoor

CVE-2024-3094, discovered by Andres Freund, March 2024



PART III

Dependency Taxonomy

You already sort dependencies this way in your head. The point isn't the categories, it's that you've never had to defend the count.

Where a dependency lives

DIRECT VS. TRANSITIVE

01 Direct: chosen explicitly by the team

02 Transitive: pulled in by the graph, still shapes security and maintenance

III · Taxonomy



Where a dependency lives

RUNTIME VS. BUILD-TIME

- 01 Build tools**
- 02 Package managers**
- 03 Plugins**
- 04 CI actions**
- 05 Deployment tooling**

III · Taxonomy



Behavioral dependencies

01 Data formats

02 Protocols

03 Defaults

04 Performance assumptions

05 Operational timing

06 Undocumented behavior

These become dependencies without a single import statement.

Dependencies by coupling

01 Compile time

02 Binary

03 Runtime

04 Protocol

05 Data format

06 Behavioral

07 Operational

One dependency can be direct, runtime critical, externally maintained, central, behaviorally coupled, and hard to replace. All at once.

Who is behind it

PEOPLE

- 01 Maintainers
- 02 Reviewers
- 03 Release managers
- 04 Vendors
- 05 Support teams
- 06 Internal experts

OWNERSHIP

- 01 Internal
- 02 Commercial
- 03 Vendor controlled
- 04 Platform controlled
- 05 Open source
- 06 Community maintained
- 07 Uncertain or abandoned

Dependencies by replaceability

- 01 Easily replaceable
- 02 Replaceable with migration
- 03 Deeply embedded
- 04 Effectively irreplaceable

Design for ease of extension and contraction. Assume today's choice will need to change.

PART IV

Dependency Principal

Every dependency creates accumulated future work. It behaves like debt, with its own interest.

The initial benefit

From concrete capability to business outcome.

01 Capability

02 Speed

03 Reduced implementation effort

04 Specialist expertise

05 Interoperability

06 Reduced time to market

The deferred obligation

In the order you actually encounter them.

01 Evaluation

02 Integration

03 Testing

04 Documentation

05 Monitoring

06 Vulnerability response

07 Upgrades

08 Compatibility work

09 Licensing review

10 Operations

11 Eventual replacement

$$\begin{aligned} &\text{Net dependency value} \\ &= \\ &\text{benefit over the product lifetime} \\ &- \\ &\text{total dependency obligation.} \end{aligned}$$

A decision model. Not a formula to solve.

Dependency principal.

**The accumulated future work created by
relying on an external component.**

Not an industry term. A useful frame for this talk.



Shipping first time code is like going into debt.



Ward Cunningham

The WyCash Portfolio Management System, OOPSLA 1992 Experience Report

The useful distinction isn't between debt or non-debt, but between prudent and reckless debt.

Martin Fowler

TechnicalDebtQuadrant, martinfowler.com, 2009

Interest on dependency principal

From routine to structural.

- 01 Recurring patching
- 02 Compatibility work
- 03 Emergency upgrades
- 04 Operational workarounds
- 05 Documentation gaps
- 06 Loss of architectural options

A prudent dependency can still create interest. That doesn't make the original choice wrong.

The cost curve

In the order you pay them.

- 01 Selection
- 02 Integration
- 03 Retention
- 04 Change
- 05 Ecosystem
- 06 Exit



The adoption decision and the exit decision are not symmetrical.

**This is genuinely hard to do,
for every dependency, every time.**

That is not the standard. The standard is that the decision was made on purpose, not by default.

PART V

Calculating Benefit and Risk

Benefit is comparative

From most control to least effort.

- 01 Build it internally**
- 02 Reuse existing capability**
- 03 Select another dependency**
- 04 Buy a commercial component**
- 05 Accept the capability gap**

Risk dimensions

CATEGORY	EXAMPLE
Security	Vulnerability, malicious release, compromised maintainer
Quality	Regression, poor test coverage, undocumented behavior
Availability	Project disappears, registry outage
Compatibility	Breaking API or runtime change
Lifecycle	Deprecation, abandonment, end of support
Legal	License incompatibility or change
Economic	Expensive commercial renewal or migration
Organizational	Maintainer concentration, unclear ownership
Regulatory	Inability to produce evidence or communicate
Strategic	Dependency constrains architecture or vendor choice

Security is the most tooled category here. It is also the smallest part of the picture.

A decision record, not a score

- 01 Alternatives considered
- 02 Assumptions made
- 03 Owner named
- 04 Expected lifetime
- 05 Support model
- 06 Exit plan

Not a single score. A record someone else could actually audit.

PART VI

Time and Lifecycle

**Most of our lifecycle diagrams
end at release.**

What if that's exactly where the real obligation starts?

**A small number of central dependencies
can dominate the risk profile.**

The burden grows heaviest exactly when the original team, context, and documentation
are already gone.

What remains possible

From lightest touch to full removal.

- 01 Update it**
- 02 Substitute it**
- 03 Isolate it**
- 04 Remove it**

PART VII

Responsibility and Accountability

The component came from somewhere else.

The accountability for using it comes home.

The accountability graph

In the order an incident would ask them.

01 Why it exists

02 Who selected it

03 Who owns it

04 Who monitors it

05 Who patches it

06 Who approves changes

07 Who communicates issues

08 How it can be replaced

One logging utility, three hops deep, that almost nobody had deliberately chosen, triggered a global emergency patch cycle overnight.

Log4Shell

CVE-2021-44228, December 2021. Verify exact scope before presenting.

Open source as shared infrastructure

People first, funding last, though funding is often the real constraint.

01 Maintainers

02 Reviewers

03 Release engineering

04 Infrastructure

05 Security response

06 Documentation

07 Succession planning

08 Funding



Whose time and labor represent a depletable resource.

Johan Linåker, Georg J. P. Link, and Kevin Lumbard

Sustaining Maintenance Labor for Healthy Open Source Software Projects through Human Infrastructure, 2024



**A business depends on a project.
It funds no code, no testing, no maintainers.**

Customer? Beneficiary? Or free rider? Support should scale with how much the business actually depends on it.

PART VIII

The CRA and Product Governance

An SBOM shows what is present.
It does not show who is responsible,
or what happens next.

**The code came from somewhere else.
Whose job is it now?**

CRA and lifecycle responsibility

Across the product's whole lifecycle.

- 01 Composition**
- 02 Vulnerability handling**
- 03 Security updates**
- 04 Technical documentation**
- 05 Support periods**
- 06 Communication**

Manufacturers shall exercise due diligence when integrating components sourced from third parties.

Regulation (EU) 2024/2847

Cyber Resilience Act, Article 13, paragraph 5

PART IX

AI as the Final Chapter

**Agents can recommend, add, remove,
update, configure, and invoke
dependencies.**

**Agents can expand or modify the
dependency graph faster than most
review processes were designed to handle.**

THE EVIDENCE

Agent authored pull requests showed a net negative security impact overall.

117,062

In 86.58% of vulnerable agent selections, a patched version was already available. Agent-introduced vulnerable choices also more frequently required a major-version upgrade to remediate, suggesting greater compatibility and migration cost

dependency changes studied across seven ecosystems. Net security impact: minus 98 for agents versus plus 1,316 for humans. Agents picked a known vulnerable version at pull request time in 2.46 percent of cases studied.

Agent-authored changes: 924 known-vulnerable dependency selections introduced; 826 vulnerabilities fixed → **net impact: -98.**

Human-authored changes: 659 introduced; 1,975 fixed → **net impact: +1,316.**

What agent evaluation should include

- 01 Reuse discipline**
- 02 Avoidance of unnecessary additions**
- 03 Safe version selection**
- 04 Policy compliance**
- 05 Remediation burden**

The new control question

- 01** Who or what introduced this dependency?
- 02** What evidence justified it?
- 03** Can the boundary change without review?

PART X

Conclusion

The dependency decision record

In the order you would actually decide.

01 Capability

02 Alternatives

03 Benefit

04 Risk

05 Owner

06 Lifecycle

07 Support model

08 Evidence

09 Exit strategy

Dependencies are valuable architectural commitments

when their obligations are visible, owned, and affordable.

**What are we willing to maintain,
support, explain, and eventually
replace?**

Thank you.

Ixchel Ruiz ixchel.ruiz@karakun.com



www.lunarbaboon.com

linkedin.com/in/ixchelruiz

[@ixchelruiz@mastodon.social](mailto:ixchelruiz@mastodon.social)

github.com/ixchelruiz



APPENDIX

Evidence and Tools

Supporting standards and tools

From one repository to the whole ecosystem.

01 OpenSSF Scorecard

02 SPDX

03 CycloneDX

04 SLSA

05 NIST SSDF

06 ENISA guidance

Scorecard describes a repository. None of these tell you the dependency will still exist in five years.