CHAOSS

A Complex Web of Open Source Software Dependencies and Risk

Overarching work of CHAOSS Risk Working group

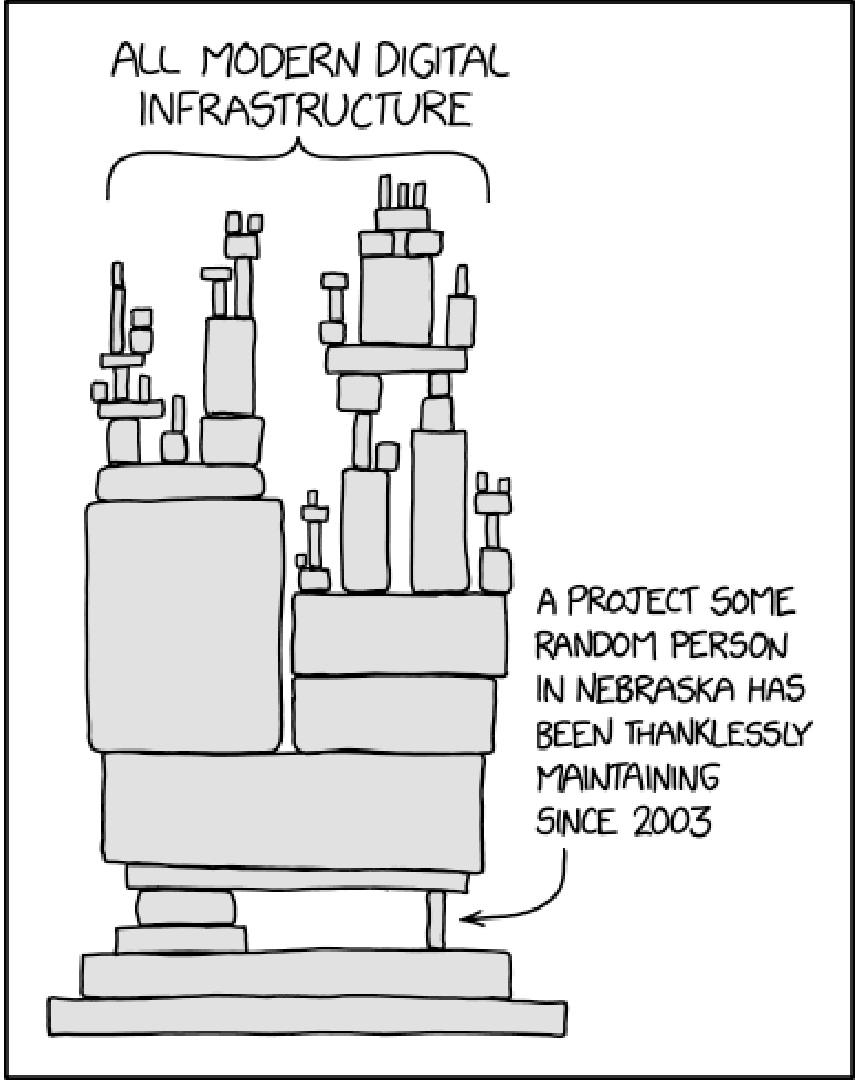
Sean Goggins with the CHAOSS Risk Working Group (Duane O’Brien, Sophia Vargas, David A. Wheeler, Arfon Smith, Kate Stewart, Michael Scovetta)

Overarching Themes Driving Dependency Concerns

1. Is my project secure enough?
2. Safe enough?
3. How do I measure this for my dependencies because they are increasingly dependent on other projects?
4. Biggest thing not being looked at: Can I can use unsafe, or less than secure component, and have secure result?
   a. How do I build a trustworthy machine without having trustworthy results?
   b. As we go from 95-97% of an application being third party dependencies, what do I control?
5. How to design so that a mistake is not the end of the world
6. How do I measure my dependencies so I can enable more rapid updates.

XKCD Summary of Dependencies

CHAOSS


ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

Categories of Risk and Who is Affected

Risk Category	**Indications of Higher Risk**	**Stakeholders With Higher Risk Exposure**
Licensing	1. Project license inconsistent with organization legal guidelines 2. Absence of Project License 3. File level licensing differences requiring analysis 4. Use of non OSI approved open source licenses	1. Technology firms producing software or selling services 2. Heavy open source consumers concerned about exposing internal software as open source
Safety Critical Systems	1. Incomplete Test Coverage 2. Runtime versus development time dependency management ambiguity 3. Absence of a software bill of materials	1. Human safety related software 2. Organizations with valuable PII
Dependencies	1. No systematic approach for dependency awareness 2. Lack of awareness of highest risk (most used) across a project portfolio. 3. Absence of a software bill of materials	1. Open source program offices 2. Technology firms producing software or selling services 3. Organizations consuming open source software without knowledge of the impact of dependencies on overall risk
Open Source Projects and Repositories	1. Low activity levels 2. Low number of maintainers and contributors 3. Long term unclosed issues and pull requests 4. Identified and unidentified project vulnerabilities	1. Everyone
Open Source Software Consumers. . .	1. Non-engagement with communities producing critical software 2. Absence of systematic maintenance of internally developed applications	1. Everyone
Enterprise Security	1. Incomplete awareness of dependency chains in deployed applications. 2. Absence of network layer security impeding bad actors, trust, identity	1. Organizations consuming open source software without knowledge of the impact of dependencies on overall risk
Sustainability	1. Low activity levels 2. Low number of maintainers and contributors	
Compliance		

CHAOSS

Resources to Address Dependency Risk

Resource	**Link**	**Description**
"Open Source Insights" by Google	https://deps.dev/	Searchable package dependencies
OSSF Scorecard	https://github.com/ossf/scorecard	Scores on 10 key items
OWASP Dependency Check	https://owasp.org/www-project-dependency-check/	Identify known vulnerabilities
Proactive Error Detection in Software	https://github.com/google/oss-fuzz	C/C++, Rust, Go, Python and Java/JVM code supported.
High Severity Vulnerability Detection	https://github.com/google/tsunami-security-scanner	Network security scanner
Kubernetes focused supply chain security	https://github.com/grafeas/kritis	Kubernetes focused
Verification from source to binary	https://reproducible-builds.org/	A myriad of reproducibility tools.
Securing Critical Projects OSSF Working Group	https://docs.google.com/document/d/1MIXxadtWsaROpFcJnBtYnQPoyzTCIDhd0IGV8PIV0mQ/edit	Managing Threats in OSS
Preventing Supply Chain Attacks	https://www.linuxfoundation.org/en/blog/preventing-supply-chain-attacks-like-solarwinds/	Enterprise Level Hardening in wake of the Solar Winds Attack
National Vulnerabilities Database	https://nvd.nist.gov/vuln/full-listing/2021/1 https://nvd.nist.gov/vuln/data-feeds#JSON_FEED	Human Readable
Libyears	https://github.com/nasirhjafri/libyear https://github.com/sesh/piprot	Tools for Libyear
Census II	https://drive.google.com/file/d/1zyAdbftGhSUiddh1she3X_MDlKXDSlu5/view?usp=sharing	Annual LF Census
2021 State of Open Source Vulnerabilities	https://drive.google.com/file/d/1BwJD3eqynwSms5b9WxzzHrzp-YRXMbLv/view?usp=sharing	A State of Vulnerabilities Report.

Overarching Themes Driving Dependency Concerns

1. Is my project secure enough?
2. Safe enough?
3. How do I measure this for my dependencies because they are increasingly dependent on other projects?
4. Biggest thing not being looked at: Can I can use unsafe, or less than secure component, and have secure result?
    a. How do I build a trustworthy machine without having trustworthy results?
    b. As we go from 95-97% of an application being third party dependencies, what do I control?
5. How to design so that a mistake is not the end of the world
6. How do I measure my dependencies so I can enable more rapid updates.

Dependency Metrics

CHAOSS

- What are the indicators of dependency risk?
- How can we quantify those risks in a meaningful way?
- Results of having those measurements
- Temporal Analysis: How important is knowing how dependencies evolve over time?
- What is the value of a particular dependency measurement?

CHAOSS

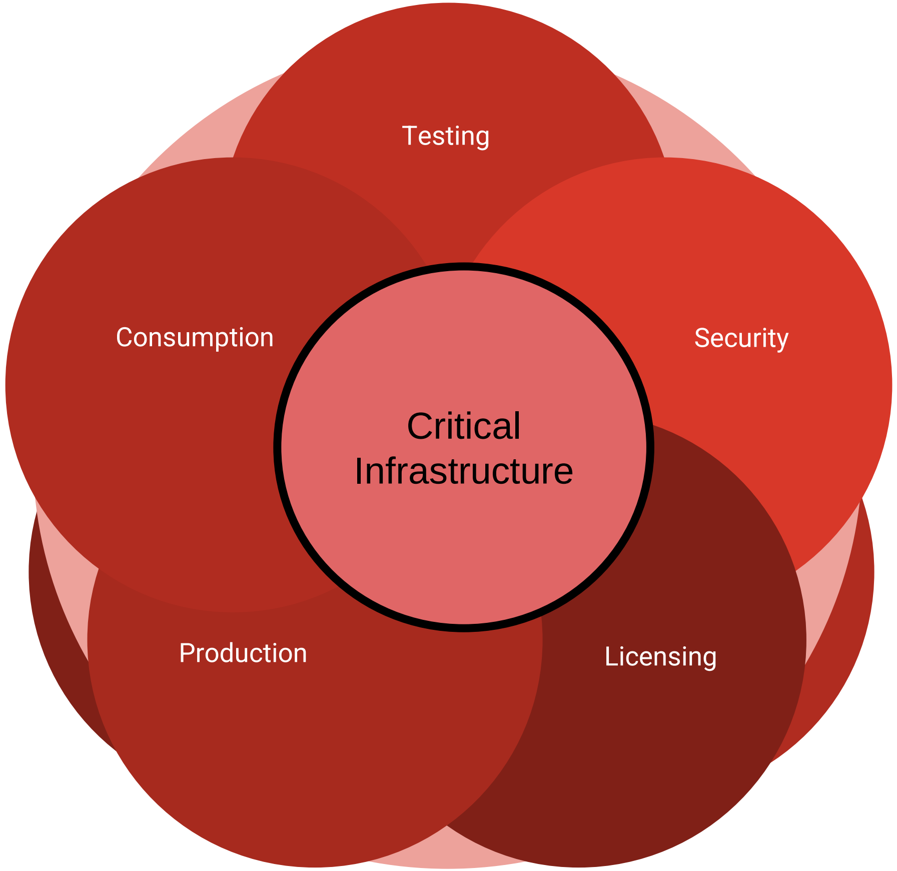
Dependency Metrics: Libyear

The CHAOSS PROCESS

- Goal – What is our goal?
  - Understanding the scope of dependencies in OSS Projects.
  - Identifying “higher risk dependencies”
  - Focus Area in CHAOSS: Risk
- Question:
  - What is the age of the project’s dependencies compared to current stable releases?
- Metrics: Libyear

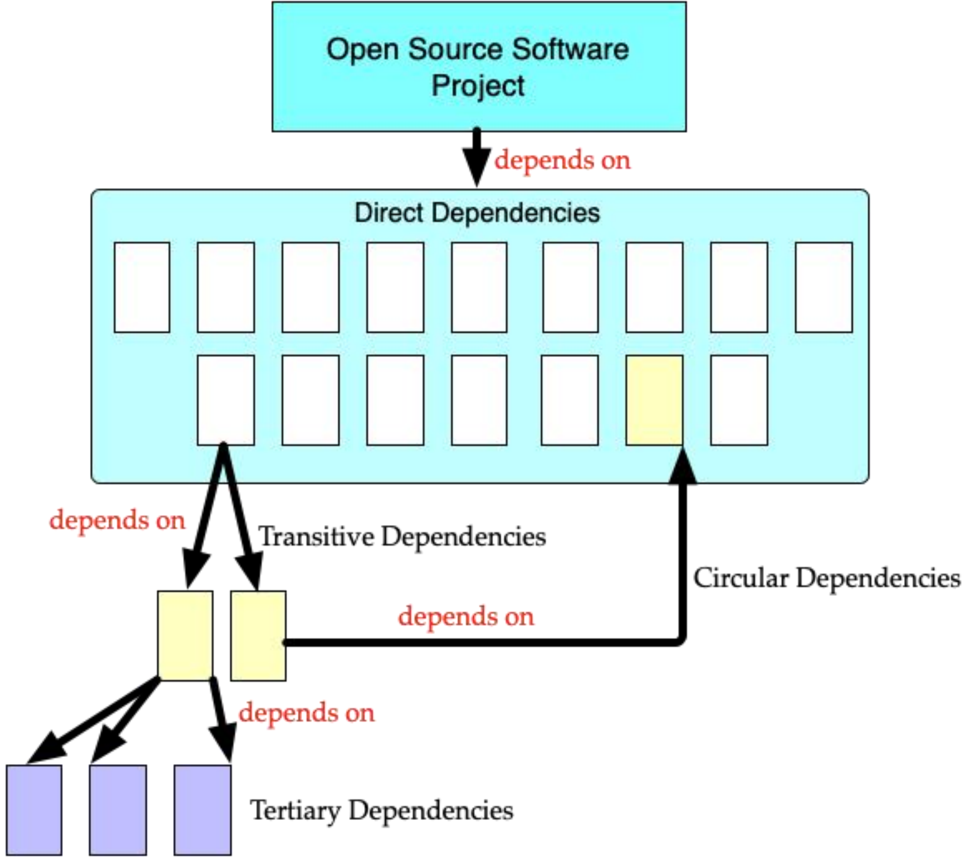
Dependencies, Risks, Vulnerabilities

1. People
   a. Tribal Knowledge
   b. Historical Knowledge
   c. Expertise in domain, security, safety, privacy, etc.
   d. Multiplicity & Diversity of people
2. Money
   a. Investment
   b. What's it cost to operate
3. Maintainability
   a. Are you keeping track of the project;
   b. Is the project being maintained
4. Test: Test Coverage
5. Dependability: Fit for purpose
6. Provenance
   a. Export restrictions



Minimum Viable Metrics (MVM's)

- Repository dependency enumeration (Upstream: Projects that my project depends on)
- Dependency sustainability risk (possibly an accumulation of Evolution, Common and Value working group metrics)
- Dependency (range?): How many times is a single dependency is referenced (in a given ecosystem, like an OSPO?)
- Libyears for projects/libraries my project depends on (total, average)
- Enumeration of known vulnerabilities for my project's downstream dependencies
- Possibly enabling OSSF Scorecards as both a metric and a tool https://github.com/ossf/scorecard
- Matrix: If there is a known vulnerability, and the project is not active, the combination of these two factors indicate greater risk.



Health and Sustainability Metrics Define What we Measure in a Repository

Repo level focus

Ecosystem level focus

- 2016: No **<u>Common, Shared</u>** Definitions
- 2016: Artisan Activity Metrics

- 2021: 50+ **Defined** Metrics
- 2021: Augur: Examples of Stories about Health and Sustainability

- 2026: Health and Sustainability Metrics Widely Available
- 2026: Tools Looking Across Projects and Ecosystems.

Trace data is a building blocks for nearly all measures of open source project health [9], [10], [11], making the collection and analysis of data related to the construction of open source at once essential for representing open source project health

<u>Activity Metrics: Repository Focus</u>

- Sufficient scale
- Project culture
- Process quality
- Product quality
- Contextualized risk
- License risk
- Corporatization and access to resources.

CHAOSS

Overarching Themes Driving Dependency Concerns

1. Is my project secure enough?
2. Safe enough?
3. How do I measure this for my dependencies because they are increasingly dependent on other projects?
4. Biggest thing not being looked at: Can I can use unsafe, or less than secure component, and have secure result?
    a. How do I build a trustworthy machine without having trustworthy results?
    b. As we go from 95-97% of an application being third party dependencies, what do I control?
5. How to design so that a mistake is not the end of the world
6. How do I measure my dependencies so I can enable more rapid updates.

Thank You