Understanding the response to open-source dependency abandonment in the npm ecosystem

Presenter: Kevin Jie

2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)

Background and Motivation

- Research on **Open-Source Dependency Abandonment**
  - Particularly the npm ecosystem
- **Core Issues**
  - Widely-used packages become unmaintained
  - Impact on open source projects which are "digital infrastructure"
  - Security risk and lack of support
- **Existing Research Gap**
  - Prior work mostly focused on:
    - Understanding and predicting why contributors or projects disengage/abandon (e.g., burnout, onboarding, etc.).
    - Dependency management more broadly (versioning, breaking changes, security vulnerabilities, technical lag).
- These work did not **quantify how common abandonment is or how developers actually react at scale**

Research Questions

- *How Big is the Abandonment Problem?*
  - **RQ1a: How prevalent** is abandonment among widely-used npm packages?
  - **RQ1b: How many** open source projects are **exposed** to abandoned dependencies?
- *What are Downstream Developers Doing About Abandoned Dependencies?*
  - **RQ2a: How often and how fas**t do dependent projects remove abandoned open source dependencies?
  - **RQ2b:** How does this **compare to how projects update** dependencies in general?
  - **RQ2c:** How does this **compare to how projects update dependencies with security vulnerability** patches?
  - **RQ3:** What **dependent project characteristics** are associated with removing abandoned dependencies?
- *What Can Maintainers Do?*
  - **RQ4:** How does **announcing the abandonment status** of a package **impact how fast** dependent projects remove the abandoned dependency?

Experiment Setup

- **Time window**:
  - January 2015 – December 2020 for identifying events (abandonment, updates, security patches).
  - Developer reactions are observed up to September 1, 2023.
- **Data sources:**
  - npm registry metadata (downloads, releases).
  - GitHub repositories (commits, issues, events).
  - World of Code (WoC) for large-scale dependency extraction from package.json snapshots.
- **Result:**
  - Start with 1,063,835 npm packages.
  - Final dataset: 28,100 widely-used npm packages.

Methodology

- **Detect Abandoned Packages**:
  - Explicit-notice abandonment
  - Activity-based abandonment
- **Identify Exposed Dependent Projects:**
- **Collect Events:**
  - Regular dependency updates
  - Security patch updates
- **Detect Dependent Reactions by When:**
  - Removed an abandoned dependency (by editing package.json).
  - Updated to the new version.
  - Updated to the security-patched version.

Result

- **RQ1a – How prevalent is abandonment among widely-used npm packages?**
  - Finding:
    - Abandonment is **common** even among widely-used packages.
  - Key results:
    - Out of 28,100 widely-used npm packages, 4,108 (about 14.6%) were abandoned during the study window.
    - **Explicit notices of abandonment are rare**: fewer than 5% of abandoned packages clearly announce their EOL status.

- Thus, nearly all abandoned packages appear inactive without maintainers ever informing users.

- Implication:
  - Downstream projects often cannot easily determine that a critical dependency has been abandoned.

Result

- **RQ1b – How many dependent projects are exposed to abandonment?**
  - Key results:
    - **About 283, 207 projects in GitHub** are a direct dependent of the abandoned projects
    - **About 78, 023 projects in GitHub** are still active after

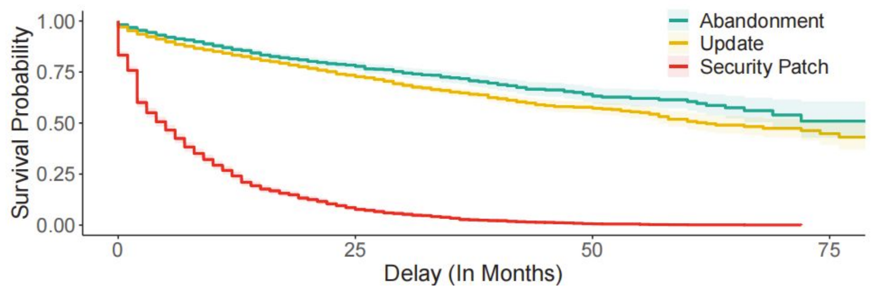
- Implication:
  - Abandonment creates exposure for a substantial share of the ecosystem.

Result

- **RQ2 – How often and how quickly do dependent projects remove abandoned dependencies?**
  - Finding:
    - Developers **rarely and slowly** react to abandonment—even slower than for routine maintenance events.
  - Key results:
    - Only **18.1% of active dependent projects ever removed** the abandoned dependency.
    - Median time to removal **exceeds 3.5 years**, based on survival modeling.

- Developers respond more slowly to abandonment than to:
    - Regular dependency updates (faster and more frequent).
    - Security patch updates, which receive the fastest reactions.

Result



Result

- **RQ3 – What project-level characteristics predict removal of abandoned dependencies?**

- Key results (from regression modeling):
  - Projects are significantly more likely to remove abandoned dependencies when they:
    - Have a **larger development team** (bus factor > 1).
    - Have **higher maintainer activity** after the abandonment date.
    - Are **younger** projects at the time of abandonment.
    - Have **higher dependency update rates** more generally
  - Other factors such as popularity, number of stars, or application domain are not strong predictors.

- Implication:
  - Only well-maintained or actively developed projects tend to react; many others leave abandoned dependencies in place indefinitely.

Result

- **RQ4 – Does explicit notice of abandonment trigger faster dependent reactions?**
- Finding:
  - **Yes**—explicit abandonment notices significantly accelerate downstream removal.
- Key results:
  - Projects depending on packages with an explicit EOL notice removed the dependency at approximately **1.58× faster** of those with silent abandonment.
    - explicit notices lead to markedly faster reactions early in the post-abandonment timeline.
  - However, explicit notices do not guarantee widespread remediation—overall removal rates remain low.

- Implication:
  - Maintainer signaling is a powerful intervention: even simple EOL notices meaningfully improve downstream risk awareness and action.

Positive Points

- **Rigorous, multi-layered empirical methodology**
  - The study combines:
    - precise heuristics for abandonment detection,
    - large-scale dependency graph extraction,
    - survival analysis for temporal modeling,
    - logistic regression for behavioral predictors.
  - This multi-method approach is unusually thorough for ecosystem-level research and increases the internal validity of the findings.
- **Strong motivation and insight-focus analysis**
  - The scope is generalized to the entire Github ecosystem
  - Analysis on the findings is insightful

Negative Points

- **Conservative heuristics may underestimate abandonment prevalence**
    - “Soft-abandoned” packages are missing in the scope. This limitation is acknowledged
- **Scope limited to direct dependencies**
    - Transitive exposure is ignored
    - Given that modern JavaScript applications often pull in hundreds of transitive packages
    - This restricts the generalizability of the practical recommendations.

Future Work

- Build an automated abandonment-detection service
    - detect and surface abandonment signals,
    - rank abandoned dependencies by risk or ecosystem impact,
    - provide migration suggestions or recommended replacements,
    - notify maintainers only when the abandoned package is relevant to their codebase.

Rating

- 5/5, I like how the paper’s organization, data processing, analysis, and insigt

Future Work

- Build an automated abandonment-detection service
    - detect and surface abandonment signals,
    - rank abandoned dependencies by risk or ecosystem impact,
    - provide migration suggestions or recommended replacements,
    - notify maintainers only when the abandoned package is relevant to their codebase.

Discussion

- (1) How should abandonment detection be integrated into current existing tooling?
  - Could ecosystems adopt standard abandonment-metadata formats, or integrate abandonment detection directly into package managers?

- (2) Should the community develop norms or governance structures for “responsible sunsetting”?
  - What would an ecosystem-wide policy look like for notifying dependents, recommending alternatives, or ensuring long-term sustainability?

- (3) How could AI come into play for the issue?
  - SWE-Agents are very helpful

Reference

[1] Miller, C. et al. 2025. Understanding the Response to Open-Source Dependency Abandonment in the npm Ecosystem. 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE). (Apr. 2025), 2355–2367. DOI:https://doi.org/10.1109/icse55347.2025.00004.