INTERVIEW-BERICHT

LUKAS DANIEL KLAUSNER

1. Metadaten

Im Zuge des netidee-Projekts 7399 „Extending the Critical Open-Source Software Database“ wurden 22 semistrukturierte Interviews mit Gesprächspartner:innen quer durch das OSS-Ökosystem geführt. Primär handelte es sich bei den Interviewten um Software-Entwickler:innen, es waren aber auch einige Menschen aus benachbarten Bereichen (z. B. unterstützende Rollen in Firmen oder Organisationen mit starkem OSS-Bezug) darunter.

Die Interviews hatten drei Themen als Schwerpunkte:

- Definitionen und Verständnisse von OSS-Qualität und -Projekt„gesundheit“
- Konzepte und Charakteristika von Kritikalität
- Bedürfnisse, Erwartungen, Feature-Requests etc. bzgl. CrOSSD

Alle Interviews wurden aufgezeichnet und unter Zuhilfenahme von Whisper (konkret: noScribe[1]) transkribiert. Zum Zeitpunkt der Berichtslegung bei Projektende ist die Auswertung (induktive, offene Kodierung und nachfolgende Analyse), die in zwei wissenschaftliche Publikationen einfließen wird, noch im Gange; dieses Dokument stellt einen kurzen Vorabbericht dazu dar. Die Interviews dauerten zwischen 31 min und 83 min; Mittelwert und Median waren (zufällig) beide 54 min.

Mögliche Interviewpartner:innen wurden primär über persönliche Netzwerke und offene Aufrufe auf diversen sozialen Medien erreicht; besonderes Augenmerk wurde dabei darauf gelegt, auch nichtmännliche Gesprächspartner:innen zu finden. Kriterium für die Beendigung der Suche nach weiteren Interviewpartner:innen war das Erreichen der theoretischen Sättigung (d. h. sinngemäß in einfachen Worten, wenn keine neuen Erkenntnisse mehr in den zuletzt geführten Interview hinzukamen).

Insgesamt wurde eine zufriedenstellende Diversität erreicht, auch durch die Verteilung des Aufrufes in verschiedenen Untercommunitys; nach wie vor im Sample unterrepräsentiert sind ältere Menschen sowie (trotz gezielter Suche) Nichtmänner. Die Verteilung der Daten (die vor Durchführung der Interviews gemeinsam mit der Einholung des informierten Konsents abgefragt wurden) war wie folgt:

- Alter: 6× 25–34, 12× 35–44, 2× 45–54, 2× 55–64
- Geschlecht: 17× männlich, 4× weiblich, 1× divers
- Karrierestatus: 9× Einsteiger:in (0–10 Jahre), 7× fortgeschritten (10–20 Jahre), 6× erfahren (>20 Jahre)
- Hauptarbeitsumfeld: 3× selbstständig, 4× Kleinunternehmen (<10 MA), 6× Mittelunternehmen (10 − 100 MA), 8× Großunternehmen (>100 MA), 1× selbstständig *und* Mittelunternehmen
- OSS-Erfahrung: 1× 0–4 Jahre, 2× 5–9 Jahre, 8× 10–14 Jahre, 2× 15–19 Jahre, 9× >20 Jahre

[1] https://github.com/kaixxx/noScribe

Die folgenden drei Abschnitte fassen die Erkenntnisse zu den drei Schwerpunkten der Interviews kurz zusammen.

2. Qualitätskriterien

In den Interviews zeichnete sich ein vielschichtiges Bild bzgl. der Qualitätsverständnisse von Open-Source-Projekten ab. Zwar wurde die Funktionalität als grundlegend angesehen, doch reichte sie selten aus. Insgesamt waren die am häufigsten genannten Faktoren für Projektgesundheit die von uns vorab erwarteten: Beliebtheit/Verbreitung der Nutzung, Dokumentationsqualität, Anzahl der Contributors und Aktivität. Es gab darüber hinaus aber einige Varianz und einige interessante Outlier in den Daten, die es noch näher zu analysieren gilt.

Die Befragten betonten insbesondere in vielen Interviews die Bedeutung einer gesunden Community als Frühindikator für die langfristige Tragfähigkeit eines Projekts. Dazu gehörten Faktoren wie die Reaktionsfähigkeit der Maintainer:innen, die Inklusivität der Mitwirkenden-Community und das Vorhandensein eines Verhaltenskodex, der aktiv durchgesetzt wurde. Über die Community hinaus wurden häufig Metriken zur Codequalität wie Testabdeckung, Ergebnisse statischer Analysen und die Einhaltung von Codierungsstandards genannt, wobei die Gewichtung solcher Aspekte sehr stark schwankte. Mehrere Befragte warnten explizit davor, sich ausschließlich auf automatisierte Metriken zu verlassen, und plädierten für eine menschliche Überprüfung und das Verständnis der Codebasis.

Die Dokumentation wurde durchweg als kritischer, aber oft vernachlässigter Bereich identifiziert, der sich sowohl auf die Benutzungsfreundlichkeit als auch auf die Einbindung neuer Mitwirkender auswirkt. Schließlich wurde die Sicherheitslage – einschließlich Prozessen zur Meldung von Schwachstellen und proaktiven Sicherheitsaudits – zunehmend als wesentlich für den Aufbau von Vertrauen und die Gewährleistung der dauerhaften Projektgesundheit angesehen, insbesondere bei Projekten mit breiterer Akzeptanz. Die Fähigkeit, Fehler und Schwachstellen schnell und effektiv zu beheben, wurde häufig als ein wichtiges Zeichen für ein gesundes Projekt genannt.

3. Kritikalität

Die Definition von Kritikalität erwies sich als überraschend komplex; wenige der Interviewten hatten vorab eine klare Vorstellung davon und viele stellten Rückfragen, was wir denn damit meinten. Dennoch zeichneten sich gewisse Gemeinsamkeiten in den Interviews ab. Ein roter Faden war insbesondere die Frage der Dependencys und der Auswirkungen im Falle eines Ausfalls. Kritische Open-Source-Software wurde als tief in andere Systeme, Infrastrukturen oder Arbeitsabläufe eingebettet beschrieben, was bedeutet, dass ein Ausfall oder eine Kompromittierung erhebliche Folgewirkungen nach sich ziehen würde.

Mehrere Befragte unterschieden zwischen verschiedenen Arten von Kritikalität. Operative Kritikalität bezog sich auf Software, die für den täglichen Betrieb unerlässlich ist, während sich Sicherheitskritikalität auf Software bezog, die die Vertraulichkeit, Integrität oder Verfügbarkeit von Daten beeinträchtigt. Wirtschaftliche Kritikalität kristallisierte sich als ein weniger häufig genannter, aber wichtiger

Aspekt heraus – Software, die für umsatzgenerierende Prozesse von zentraler Bedeutung ist und deren Ausfall schwerwiegende Auswirkungen auf die wirtschaftliche Überlebensfähigkeit haben könnte.

Zu den Merkmalen kritischer Open-Source-Software gehörte weiters das Fehlen leicht verfügbarer Alternativen, hohe Wechselkosten und ein großflächiger Einsatz. Die Befragten hoben zudem die Bedeutung der Sicherheit der Software Supply Chain hervor – also das Verständnis der Herkunft und Integrität der Abhängigkeiten der Software. Ein wesentliches Merkmal war zuletzt auch das Potenzial für systemische Risiken – das heißt, eine Schwachstelle in einer kritischen Komponente könnte eine große Anzahl von Organisationen und Nutzer:innen betreffen.

Zusammenfassend waren die wichtigsten Kritikalitätskriterien demnach: Beliebtheit/Verbreitung der Nutzung, Auswirkungsschweregrad im Falle eines Ausfalls oder eines kritischen Fehlers, Ersetzbarkeit und Sicherheitsrelevanz.

4. Plattform-Feedback

Die Interviews zeigen, dass CrOSSD als potenziell sehr wertvoll angesehen wird und eine klare Lücke in der Open-Source-Sicherheitslandschaft schließt. Es herrschte die überwältigende Meinung vor, dass ein Tool, das die automatisierte Erkennung von Schwachstellen und die Abhängigkeitsanalyse speziell für Open-Source-Projekte erleichtert, dringend benötigt wird, insbesondere angesichts der zunehmenden Komplexität von Software-Lieferketten. Es wurden jedoch auch einige wichtige Vorbehalte und Vorschläge geäußert; einige Interviewte betrachteten ein derartiges Tool für sich selbst sogar für kaum oder gar nicht nützlich.

Die Benutzungsfreundlichkeit war ein immer wiederkehrendes Thema. Mehrere Befragte äußerten Bedenken, dass ein Tool, das sich ausschließlich auf technische Details konzentriert, für weniger technisch versierte Stakeholder schwer zu verstehen und umzusetzen sein könnte. Daher ist die Betonung einer klaren und umsetzbaren Berichterstattung von entscheidender Bedeutung. Die Integration in bestehende CI/CD-Pipelines und Entwickler-Workflows wurde als wesentlich für die Akzeptanz hervorgehoben. Die Befragten betonten, wie wichtig es sei, Reibungsverluste zu vermeiden und es Entwickler:innen zu erleichtern, CrOSSD in ihre bestehenden Prozesse zu integrieren.

Genauigkeit und die Minimierung von Fehlalarmen wurden weiters als sehr wichtig erachtet; ungenaue Ergebnisse untergraben schnell das Vertrauen. Darüber hinaus bestand großes Interesse an der Erweiterbarkeit – also der Möglichkeit, CrOSSD anzupassen, um verschiedene Sprachen, Frameworks und Vulnerability-Datenbanken zu unterstützen. Schließlich regten die Befragten an, den Fokus auf den Aufbau einer Community rund um CrOSSD zu legen, die Zusammenarbeit zu fördern und Beiträge von externen Entwickler:innen zu fördern. Ein wichtiger Punkt des Feedbacks war, Funktionen zu priorisieren, die Organisationen dabei helfen, Vulnerability-Risken zu managen, anstatt sie lediglich zu erkennen.

Insgesamt lieferten die Interviews damit eine große Bandbreite an Input für die Weiterentwicklung von CrOSSD, die wir in den kommenden Projektphasen weiterführen und einfließen lassen werden.