June 22, 2025

open nexus OS - Ein einheitliches, offenes Ökosystem für nahtlose Geräteintegration

Jenning Schäfer
jenningschaefer@gmx.de

Abstract

Das Projekt entwickelt ein sicheres, modulares Betriebssystem auf Basis des Redox OS Microkernels und der Programmiersprache Rust. Ziel ist eine moderne, offene Alternative zu Android für RISC-V-basierte Tablets und Geräte. Die Architektur vereint formale Sicherheit, strikte Prozessisolierung und Multi-Plattform-Fähigkeit. Die grafische Oberfläche basiert auf einem erweiterten COSMIC-Desktop mit Touch- und Gestenunterstützung. Neben POSIX-Kompatibilität ist auch die Ausführung von Android-Apps vorgesehen, ergänzt durch ein natives App-Ökosystem mit einer QML-ähnlichen UI-Sprache und Swift-Backend. Der Entwicklungsplan folgt einem klar strukturierten Fahrplan mit Demonstratoren, Portierungen und Entwickler-Tools. Das Projekt wird unter der Apache-2.0-Lizenz veröffentlicht, kombiniert Offenheit mit wirtschaftlicher Nutzbarkeit und setzt auf Community-Aufbau, Partnerschaften und gezielte Öffentlichkeitsarbeit. Für Infrastruktur, Vollzeitentwicklung, Community-Förderung und Marketing wird finanzielle Unterstützung durch Fördergelder, Sponsoren und Crowdfunding angestrebt. Das Projekt adressiert die wachsende Nachfrage nach vertrauenswürdiger, freier Software im RISC-V-Ökosystem und legt den Grundstein für ein nachhaltiges alternatives mobiles Betriebssystem.

Inhaltsverzeichnis

1 Einführung und Motivation

1.1 Die Vision eines offenen, einheitlichen Ökosystems

In der heutigen digitalisierten Welt gewinnen intelligente Geräte zunehmend an Bedeutung. Die „Intelligenz“ dieser Geräte resultiert jedoch nicht primär aus der bloßen Rechenleistung eines Prozessors oder der Integration großer Sprachmodelle wie ChatGPT. Vielmehr liegt sie in der Fähigkeit, relevante Funktionen kontextabhängig und automatisiert bereitzustellen – ohne manuelle Konfiguration durch den Nutzer.

Ein intelligentes System erkennt, welche Informationen und Bedienoptionen im jeweiligen Nutzungskontext erforderlich sind, und reduziert dadurch Komplexität. Intelligenz manifestiert sich auch in der Fähigkeit von Geräten, nahtlos miteinander zu interagieren – idealerweise ohne explizite Kopplung oder Konfiguration durch den Nutzer. Voraussetzung für eine solche Interaktion ist ein umfassendes, interoperables Ökosystem. Ein klassisches Beispiel ist das automatische Entsperren eines MacBooks durch das Tragen einer Apple Watch – ein Vorgang, der durch das zugrunde liegende Apple-Ökosystem ermöglicht wird.

1.2 Analyse bestehender Ökosysteme

1.2.1 Apple

Apple bietet aktuell das am weitesten integrierte digitale Ökosystem. Die automatische Kopplung von Geräten, etwa AirPods oder Apple Watches, sowie die enge Verzahnung von macOS, iOS, iPadOS und watchOS, ermöglichen eine einheitliche Nutzererfahrung. Der entscheidende Nachteil besteht jedoch in der proprietären und abgeschotteten Architektur. Dritthersteller haben kaum Möglichkeiten, eigenständig innovative Erweiterungen vorzunehmen, da offene Schnittstellen weitgehend fehlen. Damit ist das Apple-Ökosystem zwar benutzerfreundlich, jedoch weder skalierbar noch zukunftsfähig im Sinne eines offenen Technologiestandards.

1.2.2 Microsoft

Microsoft verfolgt einen stärker desktopzentrierten Ansatz. Innerhalb der Windows-Welt existieren Synchronisationsmechanismen über Microsoft-Konten, insbesondere in Kombination mit Office 365 und Azure-Diensten. Für Unternehmen besteht die Möglichkeit, über Cloud-Dienste Richtlinien für Geräte zu definieren und zentrale Verwaltungs- und Kommunikationsmechanismen aufzubauen. Die Integration mobiler Endgeräte ist hingegen fragmentiert und erfordert meist separate Lösungen von Drittanbietern. Der Aufbau sicherer mobiler Arbeitsumgebungen – etwa im Bildungsbereich oder in kleinen Unternehmen – ist mit erheblichem Entwicklungsaufwand verbunden.

1.2.3 Android

Android leidet unter zwei strukturellen Problemen: Erstens basiert das System auf einem sehr schlanken Open-Source-Kern (AOSP), was zu einer starken Fragmentierung durch unterschiedliche Hersteller führt. Ein konsistentes Nutzungserlebnis ist meist nur innerhalb einer bestimmten Gerätefamilie (z. B. Samsung) gewährleistet. Zweitens ist das Android-Ökosystem stark von Google-Diensten abhängig. Diese Abhängigkeit steht im Widerspruch zu Datenschutzanforderungen und der Idee offener, kontrollierbarer Softwareinfrastrukturen.

1.2.4 Linux

Das Linux-Ökosystem weist sowohl strukturelle Fragmentierung als auch begrenzte mobile Integration auf. Während Linux im Server- und Embedded-Bereich aufgrund seiner Stabilität, Sicherheit und Flexibilität etabliert ist, fehlt eine zentrale, standardisierte Plattform für

Endanwendergeräte. Der Mangel an einheitlichen Paketformaten, Schnittstellen und GUI-Richtlinien erschwert die Verbreitung kommerzieller Software. Darüber hinaus existiert keine durchgängige Unterstützung für leistungsfähige ARM-basierte Hardware im Desktopbereich, was die Entwicklung plattformübergreifender Ökosysteme zusätzlich erschwert.

1.3 Ziel: Ein einheitliches, quelloffenes Ökosystem

Eine zukunftsfähige „Single-Device-Experience“ erfordert die nahtlose Integration verteilter und virtueller Ressourcen in einem offenen, interoperablen Ökosystem. Dabei müssen sämtliche Schnittstellen offen und dokumentiert sein, um Innovationspotenziale für Entwickler, Unternehmen und Institutionen zu erschließen.

Das Ziel besteht in der Entwicklung einer quelloffenen, modularen Plattform, die auf einer Android-ähnlichen Architektur basiert, jedoch eine vollständige Integration von Geräten unterschiedlichster Kategorien ermöglicht – von Smart-Home-Komponenten über Wearables und mobile Geräte bis hin zu stationären Desktops und Fahrzeugen. Um Fragmentierung zu vermeiden, muss das System selbst sämtliche Grundfunktionen bereitstellen, die für ein vernetztes Nutzungserlebnis erforderlich sind. Kommerzielle Anbieter können auf dieser Basis spezifische Erweiterungen für Unternehmen und Institutionen entwickeln, ohne die Integrität des Basissystems zu gefährden.

Ein solches Modell würde signifikante Vorteile bieten:

- **Mehr Komfort** für Endnutzer durch nahtlose Interaktion zwischen Geräten
- **Geringere Kosten** für Unternehmen durch Wiederverwendbarkeit von Komponenten
- **Mehr Transparenz** und Sicherheit durch quelloffene Standards
- **Stärkere Innovationsfähigkeit** durch offene Schnittstellen und Community-getriebene Entwicklung

1.4 RISC-V als Basis für eine einheitliche Hardware-Architektur

Ein wesentlicher Erfolgsfaktor für ein offenes Ökosystem ist eine konsistente Hardware-Basis. Derzeit sind Desktop- und mobile Geräte architekturbedingt getrennt – x86 im Desktopbereich, ARM im mobilen Sektor. Zwar haben ARM-basierte SoCs (System-on-Chip) – wie Apples M-Serie – eindrucksvoll demonstriert, welche Effizienzgewinne möglich sind, jedoch fehlen quelloffene Treiber und standardisierte Schnittstellen für den breiten Einsatz im Open-Source-Kontext.

RISC-V bietet hier eine vielversprechende Alternative. Die Architektur ist offen, modular erweiterbar und bereits heute in einer Vielzahl von Anwendungsbereichen vertreten. Fortschritte im Bereich Grafikprozessoren – insbesondere durch offene Treiber von Imagination Technologies – eröffnen die Möglichkeit, eine einheitliche Plattform auf Basis von RISC-V zu entwickeln. Angesichts der aktuellen Dynamik ist davon auszugehen, dass marktfähige RISC-V-basierte Geräte rechtzeitig zur Verfügung stehen werden, wenn unsere Systementwicklung heute beginnen würde.

1.5 Huawei OpenHarmony – ein möglicher, aber nicht idealer Kandidat

Ein existierender Lösungsansatz für ein ganzheitliches Betriebssystem stellt Huaweis OpenHarmony dar. Es kombiniert einen modularen Kernel mit offener Lizenzierung, einer konsistenten Hardwarearchitektur und einem Ökosystemansatz, der über Gerätekategorien hinweg funktioniert. Dennoch bestehen wesentliche Einschränkungen: Die Open-Source-Variante ist funktional stark reduziert, was zur Fragmentierung beiträgt.

1. Es fehlen zentrale Verwaltungs- und Kontrollmechanismen für unternehmensspezifische Anwendungsfälle.
2. Die geopolitische Abhängigkeit von einem einzelnen Anbieter stellt ein Risiko dar, insbesondere im europäischen Raum, der auf digitale Souveränität und Resilienz angewiesen ist.
3. Daher ist es sinnvoller, ein vollständig unabhängiges, europäisch getragenes Open-Source-Projekt zu initiieren, das strukturell von Grund auf offen, kontrollierbar und erweiterbar konzipiert ist.

2 Technische Architektur

2.1 Microkernel-Design: Redox OS als technologische Basis

Als Fundament unseres Betriebssystems dient der Redox OS-Kernel, einem modernen Microkernel, der vollständig in der speichersicheren Programmiersprache Rust entwickelt wurde. Die Wahl von Rust ist eine bewusste Entscheidung zur Erhöhung der Systemsicherheit und Wartbarkeit: Durch das Ownership-Modell, strikte Typisierung und umfassende Compiler-Prüfungen werden zahlreiche sicherheitskritische Fehlerklassen – wie Speicherlecks, Buffer Overflows oder Use-after-Free – bereits zur Kompilierzeit ausgeschlossen.

Redox vereint konzeptionelle Elemente aus mehreren bewährten Microkernel-Designs: Von **seL4** übernimmt es die Prinzipien formaler Verifizierbarkeit und strikter Prozessisolierung, während es von **Minix 3** eine modulare und pragmatische Architektur adaptiert. Das Ergebnis ist ein minimalistisch gehaltener Kernelspace, in dem ausschließlich essentielle Aufgaben wie Scheduling, Speicherverwaltung und Interprozesskommunikation (IPC) ablaufen. Alle weiteren Systemkomponenten – Treiber, Dateisysteme und Dienste – sind strikt im Userspace angesiedelt, was Fehlertoleranz und Sicherheit erhöht.

Ein wesentlicher Vorteil von Redox OS ist seine **Architekturunabhängigkeit**: Der Kernel unterstützt bereits x86_64 und ARM, eine Portierung auf RISC-V ist im Gange. Darüber hinaus stellt Redox eine vollständige libc-Implementierung bereit, was die Ausführung und Portierung bestehender C-basierter Software signifikant erleichtert.

2.2 Hardware-Unterstützung: Fokussierung auf offene Plattformen

Die initiale Hardwareplattform für die Systementwicklung ist das **PineTab V**, ein Tablet mit dem RISC-V-basierten Allwinner D1 System-on-Chip (SoC). Dieses Gerät wurde aus mehreren strategischen Gründen gewählt:

- **Hybride Einsatzmöglichkeit** (sowohl Desktop- als auch Mobile-Szenarien)
- **Offene Dokumentation** des Hardwaredesigns
- **Aktive Entwicklergemeinschaft**, die kontinuierliche Beiträge leistet

Diese Eigenschaften machen das PineTab V zur idealen Referenzplattform für frühe Entwicklungsphasen.

Langfristig wird die Architektur jedoch bewusst **plattformunabhängig** gestaltet. Ziel ist eine flexible Portierbarkeit auf weitere RISC-V-basierte Systeme, insbesondere auf Serien wie **StarFive VisionFive** oder **SiFive HiFive**. Die Priorisierung zukünftiger Zielplattformen erfolgt adaptiv anhand technologischer Reifegrade und potenzieller Kooperationen mit Hardwareherstellern.

2.3 Benutzeroberfläche: Modularität durch COSMIC

Für das grafische Frontend wird ein angepasster Fork der Desktop-Umgebung **COSMIC von System76** verwendet. COSMIC ist vollständig in Rust geschrieben und nutzt eine moderne, modulare Architektur mit nativer **Wayland-Unterstützung**, wodurch es ideal zur Redox-Basis passt.

Vorteile dieser Wahl sind:

- **Kohärente Technologie-Stack** (Rust durchgängig vom Kernel bis zur GUI)
- **Integrierte Touch-Unterstützung**, entscheidend für Tablet- und Mobilgeräte
- **Modularität**, die Anpassungen für unterschiedliche Geräteklassen erleichtert

Der COSMIC-Fork wird unter eigenem Namen weiterentwickelt und speziell für mobile Anforderungen erweitert. Geplante Erweiterungen umfassen eine verbesserte Gestensteuerung, Energiesparmodi und eine adaptive Benutzerführung.

2.4 Software-Kompatibilität und App-Ökosystem

Das System baut auf der POSIX-Kompatibilität von Redox OS auf und integriert einen vollständigen C-Compiler. Dadurch ist ein großer Teil der bestehenden Open-Source-Software – insbesondere aus dem GNU/Linux-Ökosystem – nach Kompilierung unmittelbar einsatzfähig. Dies schafft eine produktive Ausgangslage mit sofortiger Applikationsverfügbarkeit.

Parallel wird eine Kompatibilitätsschicht zur Ausführung von Android-APKs entwickelt, voraussichtlich über eine angepasste Hardware Abstraction Layer (HAL). Damit erhalten Nutzer schon in frühen Entwicklungsphasen Zugriff auf bestehende mobile Anwendungen, ohne auf ein natives Ökosystem warten zu müssen.

Langfristiges Ziel ist jedoch der Aufbau eines eigenständigen, nativen App-Ökosystems. Zu diesem Zweck wird eine moderne Entwicklungsumgebung (IDE) geschaffen, die sich gestalterisch an Qt Designer und QML orientiert, jedoch substantielle technologische Verbesserungen bietet.

Technologischer Stack:

- **Programmiersprache: Swift** Swift wird als primäre Sprache für Applikationslogik verwendet – nicht in Anlehnung an Apple, sondern wegen ihrer klaren Syntax, zunehmenden Beliebtheit und exzellenten Interoperabilität mit Rust über LLVM.
- **UI-Sprache: QML-ähnlich, aber sicherer** Die deklarative UI-Sprache basiert konzeptuell auf QML, integriert jedoch zusätzliche Sicherheitsgarantien durch Typisierung, Sandboxing und Validierung zur Laufzeit.

Dieser Ansatz verbindet die Produktivität moderner Frameworks mit den Anforderungen an Sicherheit, Effizienz und Portabilität, die für ein zeitgemäßes Betriebssystem im mobilen und vernetzten Kontext entscheidend sind.

3 Projektplan und Meilensteine

Ziel ist ein schneller funktionaler Prototyp zur Ansprache von Entwickler:innen, Fördergebern und potenziellen Partnern – bei gleichzeitiger Berücksichtigung langfristiger Wartbarkeit, Portabilität und Skalierbarkeit. Die Priorisierung erfolgt dabei nach dem Prinzip “Proof-of-Concept first, Platform-Sustainability second”.

3.1 Phase 1 (0–6 Monate): Emulationsbasierter Prototyp (QEMU)

Zielsetzung: Aufbau einer lauffähigen Entwicklungsumgebung auf QEMU mit funktionaler grafischer Oberfläche und rudimentärer Touch-Integration.

Meilensteine:

- **RISC-V-Portierung des Redox-Kernels**
  - Anpassung und Test der HAL (Hardware Abstraction Layer) für RISC-V, basierend auf vorhandener ARM/x86-Architektur
  - Funktionstests grundlegender Gerätetreiber (RAM, Timer, Framebuffer-Grafik) im Emulator
- **Integration des COSMIC-Desktops**
  - Kompilierung für RISC-V-Zielarchitektur
  - Anpassung der Wayland-Protokollunterstützung für Emulationsumgebungen
  - Basisfunktionen: Fensterdarstellung, Maus-/Tastatureingabe
- **Tablet-Modus & UI-Umschaltung**
  - Einführung eines responsiven UI-Layouts mit Touch-Optimierungen
  - Implementierung eines dynamischen Moduswechsels zwischen Desktop- und Tablet-Interface (z. B. durch Gesten oder Bildschirmgröße)

Ergebnis:

➔ Ein vollständig emulierter Systemprototyp, der grundlegende Benutzerinteraktion und UI-Switching demonstriert – als technisches Proof-of-Concept für Stakeholder.

3.2 Phase 2 (6–12 Monate): Portierung auf reale Hardware (PineTab V)

Zielsetzung: Funktionale Reife hin zur Alternative für bestehende mobile Plattformen, mit nativem App-Framework und wachsender Entwicklerbasis.

Meilensteine:

- **Kernel- und Sicherheitserweiterungen**
  - Einführung sicherer Applikationssandboxes
  - Verbesserte Energieverwaltung (aktive & passive Energiesparmodi, adaptive CPU-Steuerung)
- **App-Entwicklungsumgebung**
  - Aufbau einer nativen IDE für UI-Entwicklung mit deklarativer Sprache (QML-inspiriert)
  - Swift als Backend-Sprache mit Anbindung an die System-APIs über LLVM-Bindings
  - Beispiel-Apps: Notizen, Webbrowser, Medienplayer
- **Community & Infrastruktur**
  - Entwicklerdokumentation, API-Referenz, Beispielcode
  - Entwicklung eines dezentralen, quelloffenen App-Store-Konzepts

Ergebnis:

➔ Ein stabiler Entwicklerprototyp mit eigenem nativen App-Ökosystem – geeignet für Community-Aufbau, Pilotprojekte und weiterführende Investitionen.

3.3 Langfristige Perspektive (ab 24 Monaten): Skalierung & Ökosystembildung

- **Hardwareseitige Ausweitung**
  - Unterstützung weiterer RISC-V-Plattformen: Smartphones, Laptops, Mini-PCs

- **Android-Kompatibilität als Übergangspfad** -Vervollständigung der Android-API-Kompatibilitätsschicht (inkl. Google Services-Funktionalität optional)
- **Eigenständiges Ökosystem**
  - Plattform mit vollständiger nativer App-Unterstützung (Rust/Swift)
  - Sicherheitsmodell für digitale Souveränität und Datenschutz
  - Integration in alternative Distributionskanäle (z. B. öffentliche Verwaltung, Bildung, Open-Source-Communities)

4 Open-Source-Strategie

Unsere Open-Source-Strategie zielt auf den Aufbau eines lebendigen, nachhaltigen und zugleich geschäftsfreundlichen Ökosystems. Im Mittelpunkt stehen ein durchdachtes Lizenzmodell, transparente Entwicklungsprozesse, aktive Community-Pflege und strategische Partnerschaften.

4.1 Lizenzmodell: Apache 2.0 für maximale Flexibilität und Rechtssicherheit

Der Kernel wird unter der **Apache-2.0-Lizenz** veröffentlicht. Diese Wahl bietet signifikante Vorteile gegenüber traditionell genutzten Open-Source-Lizenzen:

- **Rechtssicherheit durch Patentklausel** Die Apache-Lizenz schützt alle Beteiligten aktiv vor Patentstreitigkeiten – ein wesentliches Kriterium für die Nutzung durch Unternehmen.
- **Geschäftsfreundlich durch Permissivität** Im Gegensatz zu Copyleft-Lizenzen (z. B. GPL) erlaubt Apache 2.0 kommerziellen Partnern, eigene Erweiterungen geschlossen zu halten – ein zentrales Argument für Hardware-Hersteller, Systemintegratoren und OEMs.
- **Kompatibilität mit GPLv3** Für die spätere Interoperabilität mit anderen freien Softwareprojekten bleibt die rechtliche Kompatibilität mit GPLv3 gewahrt.

Mit dieser Lizenzstrategie fördern wir eine breite Akzeptanz – von der freien Entwickler:innen-Community bis hin zu Industriepartnern mit Integrationsinteressen.

4.2 Community-Aufbau & Entwicklungsinfrastruktur

Eine erfolgreiche Open-Source-Plattform steht und fällt mit ihrer Community. Unser Ziel ist ein transparenter, einladender und kollaborativer Entwicklungsprozess:

- **Duale Repository-Infrastruktur**
  - **GitHub** dient als öffentliches Portal für Quellcode, Dokumentation und Issue-Tracking.
  - **GitLab** wird interne Entwicklungsprozesse, CI/CD und Roadmap-Management eingesetzt.
- **Gezielte Rekrutierung erfahrener Entwickler:innen** Fokus auf Mitwirkende aus der Redox-OS-, Rust- und RISC-V-Community
  - Priorität auf Know-how in Low-Level-Systementwicklung, Treiberarchitektur und grafischen Toolkits
- **Mentoren- und Onboarding-Programme**
  - Einführung von „Good First Issues“, detaillierten CONTRIBUTING-Richtlinien und gezielten Code-Bounties zur Senkung der Einstiegshürde

4.3 Strategische Partnerschaften & Sichtbarkeit

Die frühzeitige Einbindung relevanter Organisationen und Multiplikatoren ist essenziell für Reichweite, Vertrauen und langfristige Skalierung:

- **Partnerschaften mit Schlüsselinstitutionen**
  - RISC-V International, OSB Alliance, Free Software Foundation Europe

  - Zusammenarbeit bei Normierung, Interoperabilität, Sicherheit und Bildungsangeboten
- **Veranstaltungsformate**
  - Organisation von Hackathons, Community-Treffen und virtuell/offenen Bounty-Programmen
  - Präsenz auf Fachkonferenzen: RISC-V Summit, FOSDEM, CCC, RustConf
- **Finanzierung und Förderung**
  - Initial durch Sponsoren und Sachmittelpartnerschaften
  - Mittel- bis langfristig durch Stiftungsgelder, EU-Förderprogramme (z. B. Horizon Europe) und gezielte Crowdfunding-Kampagnen

4.4 Marketing & Entwicklergewinnung

Die Entwickleransprache erfolgt über eine authentische, technische und langfristig gedachte Kommunikationsstrategie:

- **Zentrale Projektwebsite (open-nexus-os.io)**
  - Regelmäßige technische Blogposts, Fortschrittsberichte, Roadmap-Updates und Einblicke in Designentscheidungen
- **Social-Media-Präsenz**
  - Nutzung von Mastodon, Twitter/X und YouTube für Release-Demos, Entwicklerinterviews und Livestream-Sessions
- **Attraktive Mitmachkultur**
  - Fokus auf eine respektvolle, inklusive Community mit klaren Richtlinien und offener Governance-Struktur
  - Unterstützung neuer Entwickler:innen durch Mentorenprogramme und Community-Support

4.5 Fazit

Die vorgesehene Open-Source-Strategie vereint technische Exzellenz mit pragmatischer Lizenzierung, gezieltem Community-Aufbau und strategischer Öffentlichkeitsarbeit. Das Ergebnis ist ein offenes, kooperatives Ökosystem, das sowohl idealistisch motivierte Entwickler:innen als auch kommerzielle Partner anspricht – und damit eine nachhaltige Grundlage für die langfristige Etablierung des Systems schafft.

5 Förderbedarf und Unterstützung

Für die erfolgreiche Umsetzung des Projekts ist gezielte finanzielle und organisatorische Unterstützung erforderlich. Die Mittel sollen den Aufbau eines leistungsfähigen Systems ermöglichen, das langfristig als offene, sichere und lizenzfreie Alternative im Bereich mobiler Betriebssysteme etabliert werden kann.

5.1 Technische Infrastruktur & Hardware

Für eine stabile Entwicklungsumgebung sind folgende Ressourcen notwendig:

- **Domains, Server & CI-Systeme** Einrichtung und Betrieb von Git-Servern, Build-Systemen, Container-Registern und Dokumentationsplattformen
- **Test-Hardware** Anschaffung von Geräten wie dem PineTab V, StarFive VisionFive 2 und anderen RISC-V-Plattformen zur Sicherstellung von Treiberkompatibilität und realer Performance

5.2 Vollzeitentwicklung & Kernteam

Zur kontinuierlichen und zielgerichteten Weiterentwicklung ist der Aufbau eines dedizierten Kernteams geplant:

- **Vollzeitstellen für Schlüsselrollen**
  - Rust- und OS-Entwickler:innen mit Fokus auf Kernel, Grafikstack und Hardwareabstraktion
  - UI/UX-Entwickler:innen für die Weiterentwicklung der grafischen Oberfläche (COSMIC-Fork)
  - Build- und Tooling-Spezialist:innen für CI/CD, Paketverwaltung und Cross-Compilation
- **Vertragsstruktur**: Remote-fähige Stellen auf Projektbasis, ideal für Open-Source-erfahrene Freelancer:innen oder Research Engineers

5.3 Community-Programme

Ein aktives Ökosystem entsteht nur durch Beteiligung. Dafür sind gezielte Maßnahmen vorgesehen:

- **Bounty-Programme** Finanzielle Anreize für die Entwicklung kritischer Komponenten (Treiber, Doku, IDE-Features)
- **Workshops & Hackathons** Veranstaltungsformate zur aktiven Einbindung externer Entwickler:innen, insbesondere aus den Bereichen Rust, RISC-V und embedded Systems

5.4 Öffentlichkeitsarbeit & Reichweite

Sichtbarkeit ist zentral für Nutzer-, Entwickler- und Partnergewinnung:

- **Konferenzteilnahmen** Präsentationen und Demos auf Veranstaltungen wie RISC-V Summit, FOSDEM, CCC, RustConf
- **Multimediales Marketing** Technische Blogartikel, Videodemos, Projektvorstellungen auf Social-Media-Plattformen (z. B. Mastodon, YouTube, LinkedIn)
- **Reisekosten & Material** Präsenz vor Ort zur Netzwerkbildung und Projektverbreitung

5.5 Finanzierungsstrategie

Die Gesamtfinanzierung soll sich auf mehrere stabile Säulen stützen:

Finanzierungsquelle	**Zielsetzung**
Förderprogramme	z.B. Prototype Fund, EU Horizon, ZIM
Unternehmenssponsoring	vor allem aus der RISC-V und Embedded Branche
Community-Crowdfunding	frühe Unterstützer, Beta-Tester, Open-Source-Fans
Langfristiges Stiftungsmodell	Orientierung an Modellen wie Apache Foundation

5.6 Warum sich Investitionen lohnen

- **Ein freies, sicheres mobiles OS für RISC-V fehlt bislang** Das Projekt schließt eine strategisch wichtige Lücke im Open-Source-Ökosystem.
- **Hoher Mehrwert für Industriepartner** Durch die Apache-2.0-Lizenz ist das System lizenzfrei nutzbar und leicht anpassbar – ideal für OEMs und Forschungseinrichtungen.

- **Community-getriebenes Wachstumspotenzial** Eine offene Architektur und aktive Entwicklergemeinschaft fördern schnelle Innovationen und breite Akzeptanz.