Avis de Soutenance

Madame Edlira NANO

Informatique

Soutiendra publiquement ses travaux de thèse intitulés
Obsolescence logicielle : analyse et stratégies de remédiation

Travaux dirigés par Monsieur Aurélien TABARD

Soutenance prévue le ***lundi 22 juin 2026*** à 9h30
Lieu : Université Lyon 1, Bibliothèque Sciences - Salle de conférence au 20 avenue Gaston Berger à Villeurbanne

Composition du jury proposé

M. Aurélien TABARD	Maître de conférences	Université Lyon 1	Directeur de thèse
Mme Florence MARANINCHI	Professeure des universités	Grenoble INP	Rapporteure
Mme Aurélie BUGEAU	Professeure des universités	Université de Bordeaux	Rapporteure
Mme Stéphanie JEAN-DAUBIAS	Professeure des universités	Université Lyon 1	Examinatrice
M. Sébastien SHULZ	Chargé de recherche	CNRS Paris	Examinateur
M. Roberto DI COSMO	Professeur des universités	Université Paris Cité	Examinateur
Mme Nolwenn MAUDET	Université de Strasbourg	Invitée	

Mots-clés : obsolescence, logiciel, soutenabilité, numérique, écologie, impact environnemental

Résumé :

Cette thèse de doctorat vise à analyser l'obsolescence logicielle ainsi que des stratégies de remédiation. Notre première étude de cas porte sur l'écosystème Android, le système d'exploitation le plus utilisé au monde, où les appareils sont rarement mis à jour plus de deux ans après leur sortie. Nous avons étudié les obstacles au développement et à la maintenance d'Android à travers 12 entretiens avec des acteurs clés de l'écosystème, complétés par une étude ethnographique lors de conférences et une analyse de la littérature technique. Nous montrons comment les flux de code circulent entre les différents acteurs de l'écosystème, ce qui freine les mises à jour, et nous analysons comment ces acteurs répartissent leurs efforts de maintenance à différents niveaux afin de servir leurs intérêts stratégiques. Le manque de mises à jour se manifeste au niveau du noyau, i.e. au cœur d'Android, car le code des fabricants de téléphones et de systèmes sur puce diverge de plus en plus du code du noyau Linux d'origine. Nous montrons que Google, principal acteur de l'écosystème, tente de résoudre les problèmes de maintenance en transférant la responsabilité aux fabricants de téléphones, qui sont les moins enclins à maintenir, entraînant ainsi une fin de vie prématurée des appareils. En parallèle, nous analysons comment certains vendeurs et acteurs

alternatifs de systèmes d'exploitations libres et open-source pour mobiles, mettent en œuvre des stratégies de maintenance pour assurer une plus grande longévité des appareils. Notre seconde étude de cas porte sur la maintenance de Debian, un système d'exploitation basé sur le noyau Linux, développé et maintenu depuis plus de 30 ans par une communauté internationale, selon des principes d'open-source. Nous avons participé durant trois ans à cinq réunions communautaires, et avons mené de nombreux entretiens avec des acteurs clés de Debian. Le travail de maintenance de Debian est structuré à différents niveaux, tant techniques que sociaux : au niveau du code, des paquets, du noyau Linux et de l'infrastructure interne à la communauté. Les relations sociales entre les membres de Debian et avec les développeurs externes jouent un rôle essentiel dans le processus de maintenance. La pérennité de la communauté – résolution des conflits, prévention de l'épuisement professionnel, création d'un environnement de travail et social inclusif, fidélisation des membres et recrutement de nouveaux – apparaît comme un enjeu majeur pour le maintien des fondements sociaux. Nos résultats mettent en lumière les pratiques de développement et maintenance, et le rôle de l'infrastructure technique et sociale communautaire au service de cette maintenance. En nous appuyant sur l'analyse de ces deux systèmes d'exploitation, nous analysons comment les flux de code circulent entre les acteurs et comment se met en place, ou pas, leur maintenance, et les mécanismes menant à l'obsolescence. Les points de rupture qui apparaissent peuvent être d'ordre technique (obfuscation du code, absence de documentation, mauvaises pratiques de codage) ou socio-économique (manque de volonté de maintenir, positions de pouvoir et domination de certains acteurs clés, dépendances forcées à des composants logiciels essentiels, contrats d'exclusivité logicielle, absence de politiques publiques garantissant la pérennité du système). Nous examinons l'équilibre entre ouverture et fermeture dans le développement et la maintenance des logiciels. Nous analysons comment les stratégies de maintenance et soin à long terme se placent à la fois sur le plan technique, sur le plan social et sur le plan infrastructurel. Enfin, cette étude propose plusieurs recommandations réglementaires en matière de pratiques de codage durables, et de politiques publiques interdisant les freins et venant en soutien à la maintenance des systèmes d'exploitation.

Summary:

This PhD thesis aims to analyze software obsolescence and existing remediation strategies. Our first case study focuses on the Android ecosystem, the world's most widely used operating system (OS), where devices are rarely updated more than two years after their release. We investigate what hinders Android development and maintenance. We conducted 12 interviews with key players in the ecosystem, supplemented by conference ethnography and analysis of technical literature. We show that the way code flows are organized across the various ecosystem actors inhibits updates, and we outline how these actors locate their maintenance efforts in different places to serve their strategic interests. The lack of updates appears at the kernel level, i.e, at the core of Android builds, as the code from phone vendors and system on chip manufacturers increasingly diverges from the original Linux kernel code. We show that Google, the main actor governing the ecosystem, addresses maintenance issues by shifting responsibility towards phone vendors. However, as vendors are the least inclined actors to maintain their code, the problem persists, leading to premature end-of-life for devices and, consequently, their obsolescence. At the same time, we analyze how, driven by a concern for longevity, some vendors and alternative free open-source mobile actors are implementing remediation strategies to maintain devices. Our second case study focuses on long-term maintenance within Debian, a widely used OS based on the Linux kernel, maintained by a community organized as a non-profit, following the principles of collaborative open source development. We assisted in 5 community gatherings and conducted 12 interviews with key players in the ecosystem, supplemented by analysis of technical literature. Maintenance work in Debian is structured at different levels, both technically and socially: at the code level, at the package level, at the Linux kernel level, and at the inner-community infrastructure level. It is the social relationships

between Debian members and with external upstream developers that play an essential role in the maintenance process. Sustainability within the community: resolving conflicts, avoiding burnout, creating an inclusive work and social environment, retaining members, and attracting new ones seem to be important concerns for maintaining social foundations. Our findings highlight the role that the technical and social infrastructure developed by Debian within its community plays in maintaining a robust and sustainable operating system. But they also underscore the importance of the collective process of the community's reflection on improving it. Reflecting on the conclusions drawn by Android and Debian, we discuss the various strategies we have observed in terms of code flows between actors, and how they inhibit or facilitate maintenance. Breaking points can be technical: code obfuscation, lack of documentation, anti patterns in coding practices, but also socio-economic: positions of power and dominance of certain key actors within ecosystems, imposition of dependencies kept private on essential software, legal contracts requiring exclusive use of software, or lack of public policies ensuring longevity. We discuss the play between openness and closure in software development and maintenance, as well as the importance of open standards in building independent and resilient systems. Upstreaming and mainlining appear to be important maintenance strategies at the software development level. Social interactions and maintenance infrastructuring support the sustainability of maintenance work. The study attempts to formulate a number of recommendations for regulatory measures in order to enforce sustainable coding practices, while prohibiting the abuse of dominant positions. It also seems essential to put in place public policies that support and accompany fundamental software ecosystems such as operating systems.