FOCUS NOTE

OPEN-SOURCE TECHNOLOGIES IN THE CONTEXT OF FAST PAYMENT SYSTEMS

F4D FINANCE FOR DEVELOPMENT

PROJECT FAST
FRICTIONLESS • AFFORDABLE • SAFE • TIMELY • TRANSACTIONS

FEBRUARY 2025

WORLD BANK GROUP

FINANCE, COMPETITIVENESS & INNOVATION GLOBAL PRACTICE
Payment Systems Development Group

© 2025 International Bank for Reconstruction and Development / The World Bank
1818 H Street NW
Washington DC 20433
Telephone: 202-473-1000
Internet: www.worldbank.org

This volume is a product of the staff of the World Bank. The findings, interpretations, and conclusions expressed in this volume do not necessarily reflect the views of the Executive Directors of the World Bank or the governments they represent.

The World Bank does not guarantee the accuracy of the data included in this work. The boundaries, colors, denominations, and other information shown on any map in this work do not imply any judgment on the part of the World Bank concerning the legal status of any territory or the endorsement or acceptance of such boundaries.

RIGHTS AND PERMISSIONS
The material in this publication is subject to copyright. Because the World Bank encourages dissemination of their knowledge, this work may be reproduced, in whole or in part, for noncommercial purposes as long as full attribution is given.

CONTENTS

1 EXECUTIVE SUMMARY

Open-source software has gained significant popularity due to its zero initial cost and widespread adoption across industries. Recent studies show that open-source software appears in nearly 97 percent of codebases across different sectors. A Harvard University study estimates that re-creating widely used open-source software would cost about $4.5 billion, and up to $8.8 trillion if every firm leveraging open-source software re-created the software. These figures highlight the significant economic impact and importance of open-source software across the global software landscape.[1]

In the context of a fast payment system (FPS), the use of open-source software for core components is still at an early stage. Our research identified only one live FPS using open-source software for core functionalities. Despite the limited adoption of open-source software in FPS, interest is growing among payment system operators, particularly in initiatives such as Mojaloop. Moreover, some FPS operators may choose to use open-source software for non-core components, making it crucial for both operators and regulators to be aware of the potential impacts that open-source software may have, both positive and negative, on the overall efficiency and resilience of FPS. This note, given rising interest in open-source software for FPS, examines its use as well as relevant implementation models, risks, benefits, and costs. The World Bank remains neutral on the use of open-source software in FPS, neither endorsing nor discouraging its adoption. However, it encourages countries and FPS operators to evaluate their institutional capacity to implement open-source software aptly and manage its risks, the level of support from the open-source community, and the full cost implications beyond initial deployment. While open-source software offers advantages, such as cost savings from reduced licensing fees and the flexibility to customize systems, it also presents challenges. Institutions need significant in-house expertise for customization, maintenance, and operation and must align their long-term FPS goals with the chosen open-source solutions to ensure adaptability and scalability. Thorough cost-benefit analyses are essential to weigh the immediate benefits against potential long-term challenges, particularly around operational capacity and system enhancements.

Moreover, adopting open-source software for FPS introduces specific risks, particularly around cybersecurity and integration with existing systems. Vulnerabilities in open-source components can expose systems to a broader range of threats. While the open nature of open-source software can enhance transparency, it can also expose vulnerabilities to a wider audience, including potential attackers. As demonstrated by past security incidents, such as the Log4Shell vulnerability, unpatched or outdated open-source software components can pose significant risks to critical systems such as FPS.

Therefore, operators and regulators must be proactive in assessing these risks, ensuring that robust security measures are in place and continuously monitoring for updates and vulnerabilities. While open-source software offers potential benefits, these must be balanced carefully with operational, security, and regulatory considerations to ensure the resilience and long-term sustainability of FPS infrastructures.
The note highlights the following learnings:

- **Open source in payments is still a growing space.** There is increasing interest from operators in the benefits that open source can bring to payments, including FPS. Mojaloop is one example that operators have just begun to assess actively. However, it is key to note that some FPS do utilize open-source components, particularly for non-core software and infrastructure.
- **Consider the following key factors across the life cycle of an FPS before leveraging open source:**

  Design and conceptualization

  - ***Requirement alignment:*** Ensuring that open-source solutions meet the functional, business, and security needs of the FPS is crucial. Operators must confirm that these solutions align with current and future requirements, and open-source projects should adhere to open standards such as ISO 20022 for seamless interoperability.
  - ***Regulatory and legal implications:*** Open-source solutions must comply with local and international regulations. Institutions must also carefully navigate intellectual property issues and open-source licensing to avoid unwanted disclosure of proprietary enhancements.
  - ***Security by design:*** Incorporating secure design principles from the beginning of the project is essential to mitigate cyber risks.

  Implementation

  - ***Adaptability:*** Open-source solutions offer customization but require significant expertise for them to be tailored to institutional needs, which can introduce security vulnerabilities if not handled properly. Proper documentation is key to managing the complexity of the implementation.
  - ***Capacity:*** Institutions need strong internal technical capabilities or external support to implement open-source FPS effectively. This includes building a dedicated team of system integrators to ensure seamless customization and security updates.
  - ***Customization versus vendor lock-in:*** Open-source solutions reduce vendor lock-in, but a lack of vendors for specific components may increase costs and timelines.
  - **Secure integration:** Operators should treat open source with the same rigor as proprietary code by selecting pre-vetted components, conducting initial and ongoing vulnerability assessments using software composition analysis tools, and performing regular scans during development and build stages.

  Operation

  - ***Security:*** While open-source solutions allow for code review and collective security checks, vulnerabilities can still remain undetected for years. Regular vulnerability assessments and third-party audits are essential.
  - ***Maintenance and support:*** Institutions must ensure that they have the internal capacity to maintain and troubleshoot open-source solutions, as they do not come with the same level of vendor support as proprietary solutions.
  - ***Strength and expertise of the open-source community:*** A highly active and skilled community ensures ongoing support, timely updates, and rapid responses to security issues. However, if the community lacks sufficient expertise or is inactive, slower updates, unresolved problems, and increased risks for the FPS may be the result.
  - ***System migration:*** Open-source solutions must consider long-term migration costs, including compatibility issues and the potential loss of customizations when transitioning to new systems.

- ***Monitoring:*** It is essential to stay updated on open-source components and monitor for vulnerabilities, promptly updating or replacing unsupported components. Additionally, implementing a software bill of materials enhances transparency, documents component origins, and aids in managing vulnerabilities.

Maturity of an organization to leverage open source

- ***Operational readiness:*** Implementing open-source FPS requires robust operational and technological capabilities, and institutions need to manage potential risks. This requires a full assessment of internal capacity as well as of available external resources, such as system integrators, to ensure seamless integration, ongoing support, and effective risk mitigation.
- ***Maturity models:*** Tools such as maturity models can help operators assess and develop their ability to manage and contribute to open-source projects, enhancing strategic and operational alignment with open-source practices.

Cost implications across the life cycle of an FPS

- ***Total cost of ownership:*** Open-source solutions can reduce initial costs by eliminating licensing fees but may increase internal development and operational expenses. Analyzing the total cost of ownership, including future maintenance, security, and migration costs, is necessary.
- ***Hidden costs:*** Costs for integrating new functionalities, adhering to evolving standards, and adapting to new use cases must be considered. FPS as a digital public infrastructure increases the pressure for continual innovation and resilience.

- **Choosing open-source technologies requires embracing a long-term vision and clearly assessing the capacity of the organization to adopt them.** Institutions implementing an FPS must ensure that their long-term vision aligns with the open-source solutions they adopt, prioritizing the adaptability and scalability of the solutions, as well as their capacity to customize them, over immediate simplicity and cost savings.
- **Operators and regulators must assess the potential risks of open-source technologies not only for FPS but also for the broader payment ecosystem.** FPS and payment systems do not operate in a vacuum, and even when they do not leverage open-source software for their core components, they can still be exposed to vulnerabilities stemming from open-source software and platforms. Open-source technology can carry cyber and operational risks, and its widespread usage and integration with proprietary components can expose vulnerabilities to a broader audience. These risks necessitate continuous monitoring, proactive contingency planning, and a thorough assessment of both internal and external resources. Therefore, regulators and operators should adopt a proactive approach to reviewing the overall impact of open-source technology across FPS and the broader payment ecosystem.

2 INTRODUCTION

The World Bank has been monitoring closely the development of fast payment systems (FPS) by central banks and private players across the globe. This comprehensive study of FPS implementations has resulted in a policy toolkit designed to guide jurisdictions and regions on the likely alternatives and models that could assist them in their policy and implementation choices when they embark on their FPS journeys. The FPS Toolkit can be found at fastpayments.worldbank.org and consists of the following components:

- The main report *Considerations and Lessons for the Development and Implementation of Fast Payment Systems*
- Case studies of jurisdiction that have already implemented fast payments
- A set of short focus notes on specific technical topics related to fast payments

3 OPEN SOURCE IN THE CONTEXT OF FPS

COMPONENTS OF FPS

An FPS allows for immediate fund availability to beneficiaries and can be used 24 hours a day, seven days a week, 365 days a year. Such a system generally includes the following components to be managed by an operator or different operators: core-clearing and settlement infrastructure,[2] an application layer, and a scheme/rulebook governing relationships between participants and relevant parties.[3]

However, establishing a new national payment rail may require significant investment. FPS involve a wide range of components that, taken together, involve ongoing costs to be managed by national payment system operators. Depending on the chosen infrastructure, participant types, and supported use cases, the cost and timeline of development can vary significantly. These can vary further if the FPS is built on existing infrastructure or developed from scratch. Aside from the cost of building the system, multiple other costs accrue to the FPS, including operating, maintenance, and marketing costs. Annex A explores the specific components of an FPS in more detail.

FPS development could be funded by the central bank of a jurisdiction, by participating institutions (banks and/or non-banks), or collectively via public-private partnerships. It requires teams comprising specialists from various domains for good execution. The FPS life cycle further entails continuous enhancement and development, requiring investment throughout.

A determining cost factor is the operator's choice to self-source, insource, or outsource development of the FPS. With self-sourcing, the operator is fully responsible for all the software, hardware, and people required to implement and operate an FPS autonomously.[4] With insourcing, an operator's innate capacity is augmented by development resources either from other government agencies or from external sources, but for the duration of the development project only.

Making a decision based on such factors as cybersecurity regulation, institutional knowledge, and cost, jurisdictions can choose to implement FPS themselves, leverage vendors, or opt for open-source software. Some payment system operators, central banks, and public authorities are evaluating open-source specifications, examining the potential costs and benefits to determine the feasibility and convenience of implementing these solutions.

WHAT IS OPEN-SOURCE TECHNOLOGY, AND HOW DOES IT DIFFER FROM PROPRIETARY TECHNOLOGY?

One way to examine the key features of the open-source technology construct is to contrast it with its opposite—proprietary technology. While in both cases the technology consists entirely of software and its source code, licenses and usage permissions differ. With proprietary technology, the organization or author of a piece of software places restrictions on the use and modification of that software, governed by licensing agreements between the user and developer. Users are often required to agree to use software as explicitly provided by the developer. This means that users are dependent on the developer to add new functionality and correct the software in line with the developer's predetermined development cycle.[5]

With open-source technology, the license[6] enables users to modify, study, use, and distribute the software and its source code for any purpose.[7] In the canonical open-source model, source code is made available for developers, along with documentation and guidance on use. The upkeep and maintenance of these projects varies across codebases (the full body of source code for a given project) but, in some cases, is overseen by a foundation or central organization. Examples include the Apache Software Foundation and the Linux Foundation. This open-source construct, mediated by its governance model, has important consequences in terms of cost, security, and functionality. A distinction should be made between an open-source system software and its open-source dependencies—modules the system depends on that are built, released, and maintained by other open-source projects. The publisher of the system software itself and the publishers of its individual dependencies together are referred to as the software supply chain.

Open-source software and components are widespread in software projects, and they are leveraged as well within proprietary solutions. Prominent applications and operating systems are based on the Linux kernel, while other applications leverage key open-source software components. Indeed, several companies exist whose core ethos is to develop software from open-source platforms. One is RedHat, which builds a wide range of open-source products, including cloud infrastructure, middleware, and operating systems (box 1).

The World Bank has made use of open source in the realms of geospatial analysis and disaster resilience, as described in box 2.

Open-source software has already been made available in payments technology.[8] A preeminent example is the open-source software Mojaloop (box 3).

BOX 2 WORLD BANK INITIATIVES THAT LEVERAGE OPEN SOURCE

GOSTNets

GOSTNets is a tool developed by the Geospatial Operations Support Team at the World Bank. GOSTNets itself was designed as a convenience wrapper for network analysis using geospatial information, particularly from OpenStreetMap.

GOSTNets utilizes open source in multiple ways to enhance its network-analysis capabilities, especially through leveraging geospatial information. GOSTNets focuses primarily on using data from OpenStreetMap, which is a collaborative project to create a free editable map of the world. GOSTNets integrates with the NetworkX Python library, which is an open-source tool designed for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

OpenDRI

The Open Data for Resilience Initiative (OpenDRI), another initiative by the World Bank, employs open-source tools to bolster resilience against natural hazards and the effects of climate change. OpenDRI uses GeoNode, an open-source data-sharing platform that facilitates public access to vital risk information. This platform is instrumental in managing, analyzing, and storing crucial data for informed planning and policy making. Further-

BOX 1 USAGE OF OPEN-SOURCE SOFTWARE

FINOS reports that open-source GitHub repositories with financial service company commits rose by 43 percent between 2021 and 2022.[9] Examples are a user-interface project by J.P. Morgan, a high-performance data store by the Man Group, and others, including applications ranging from user-interface toolkits to financial infrastructure code. Further, the cloud-native platform Kubernetes, which many companies use, is open source. Companies such as Google and Microsoft go a step further and actively participate in the open-source community.[10,11] Google, among others, has an entire team dedicated to maintaining open-source projects that are crucial to the entire industry, not just itself. Other companies, such as Razorpay (an Indian payments company), MongoDB, GitHub, and Databricks, have oriented themselves from their basic structure to support open-source software and regularly release their own source code to enable collaboration and extensibility.[12]

BOX 2, *continued*

more, OpenDRI leverages InaSAFE, an open-source software that amalgamates scientific data, community input, and local government information to forecast the potential impacts of disaster events.

Sources: https://opendri.org/about/, https://github.com/worldbank/GOSTnets

BOX 3 FPS SPECIFICATIONS DEVELOPED BY MOJALOOP

Mojaloop, an open-source FPS, was created to address the needs of individuals typically excluded from the formal financial system. One significant motivation for making Mojaloop open source was providing an affordable and accessible alternative for jurisdictions looking to establish a national payment infrastructure.

At its core, Mojaloop provides a set of interoperability and payment functionalities that facilitate fund transfers, especially geared toward faster payments The architecture of Mojaloop is based on the principles of an open-loop system,[13] in which different financial service providers can connect to the platform and exchange transactions seamlessly. Mojaloop entails three layers: an interoperability layer, connecting bank accounts, wallets, and merchants in an open loop; a directory service layer, which provides an alias-lookup service for when accounts are masked using proxies; and a transactions-settlement layer, which makes instant and irrevocable payments possible. Integration of Mojaloop need not require leveraging the entire Mojaloop package, a user can pick and choose components.

Mojaloop incorporates various standardized message formats and protocols, such as ISO 20022 and the Interledger Protocol (ILP), to enable the secure transfer of funds and information between different systems. The use of the ILP is notable because of its decentralized design and cryptographic security elements and its easy extension into interoperability.[14] It supports a wide variety of transaction types, including fast payments in a broad range of financial service scenarios.

It is important to note that while Mojaloop provides the underlying technology and infrastructure, its adoption and implementation depend on the collaboration and participation of financial institutions, regulators, and other stakeholders in the ecosystem to establish a fully functional and inclusive digital payment network.[15] As part of this, Mojaloop offers developers full access to its codebase and actively encourages contributions from them. While Mojaloop has a team that maintains the codebase, it does rely on contributions from adopters.

Mojaloop was designed to support low- and large-scale volumes, such as those seen in developed markets, such as India. Mojaloop's design is intended for horizontal scaling, with low-cost, redundant servers to ensure data integrity even if nodes fail.

Security was a core consideration from the initial design of Mojaloop, which employs a zero-trust model, two-factor authentication system and roles-based access. In its development, a community of over 2,400 members participate at various levels of involvement.

The Mojaloop stack includes the following modular services:

- API functionality
- Central services (basic clearing and settlement functionality)
- Account-lookup service (participant-lookup service)
- Quoting service (enables fee transparency on the platform)
- Fraud risk–management services

4 A TAXONOMY OF OPEN-SOURCE TECHNOLOGY FOR FPS

As open-source specifications have begun to appear in the fast payments space, an emerging broad taxonomy has incorporated the following three types of systems:

1. Ground-up open-source systems
2. Proprietary systems, with open-source peripherals
3. Proprietary software

While most FPS are closed source, newer systems incorporate open-source elements to varying degrees. Some may be fully open source, while others incorporate open-source elements.

GROUND-UP OPEN SOURCE

A ground-up open-source fast payments project places open source at its core. This means that it need not retroactively work to publish code for the benefit of its userbase. Instead, code is sourced publicly, and system-specific developments may be shared back or at least made accessible to its userbase from the start of the project. In such a scheme, the default for any module or application produced is that it is open to the public. Certain components may be held privately within the foundation or entity that manages the project, such as fraud rules or other sensitive materials, but even these are made available when possible and as appropriate. While opening the source code to the public has its benefits, especially in terms of code review and analysis for security vulnerabilities, a ground-up open-source model also contains certain cyber risks. For example, the ability to review and assess the source code independently can offer increased security when the code is examined thoroughly and from multiple angles, but at the same time, the public availability of the programming code may allow adversaries or those interested in committing fraud to study the code for potential vulnerabilities, without revealing them.[16] Initially or over time, such a project may develop a governance structure that ensures its openness while also promoting a steady pace of development. It may choose to enable users to contribute to the codebase through a structured contribution, review, and approval process. An open-source organization such as this, as best practice, will provide documentation for its projects and components. Entities choosing to implement such a project can then take this documentation and build internal uses of that project. Some open-source organizations will provide active support and guidance in the implementation process, if requested. In other cases, implementing entities may choose to leverage internal or outsourced expertise to build on existing open-source projects. Components of an open-source solution and ecosystem are included in annex B.

PROPRIETARY, WITH OPEN-SOURCE ELEMENTS

As institutions continue to implement FPS, some have begun to explore the value of making peripheral sections of these systems open source. Generally, however, these have tended toward open-standards approaches and not genuine open source. FPS have generally been closed source by default, given the high level of effort required to

develop them and other considerations, including security. Further, the community that might participate in such an open-source project is necessarily small, given that it would be made up only of central banks and payment system operators and systems integrators that connect financial institutions to these FPS. The fact that the community is small and that there may be insufficient resources to adequately evaluate the security of open-source components, such as open-source modules, or libraries, may increase the level of cyber risk associated with such initiatives. However, established FPS have made small steps to open-source some aspects of these systems. In some cases, communication protocols have been made open for use and input by the community. Open-communications standards, especially using the Open API construct, have enabled faster and more responsive connectivity with central banks. Examples of proprietary FPS that have integrated or developed open-source solutions are included in box 4.

PROPRIETARY SOFTWARE

As mentioned previously, most fast payment implementations are closed source. In many cases, this is the default approach; for others, it simply has not made sense to use open-source specifications. For many jurisdictions, the prospect of building an FPS is challenging, given the limited level of expertise on the topic that exists within central banks. For many, it makes sense to outsource development and maintenance fully to a third-party vendor. In these cases, central banks can provide their requirements in terms of features, stability, and security and receive an FPS built and maintained by an experienced team. These projects are necessarily closed source.

Examples of proprietary FPS include systems operated by the private sector, such as the New Payments Platform in Australia, PromptPay in Thailand, and the Faster Payments Service in the United Kingdom.

There is another type of proprietary solutions—those developed by operators, such as central banks, on their own. These systems were specifically tailored to meet the needs of the local market, ensuring compliance with local regulations and standards and integrating seamlessly with the existing financial infrastructure. The systems are proprietary, and key technological components are not made available to the public. Examples of this type are the FPS built and operated by the central banks of Mexico (SPEI), Turkey (FAST), and Costa Rica (Sinpe Móvil), which have developed core com-

BOX 4 USAGE AND DEVELOPMENT OF OPEN SOURCE BY PROPRIETARY FPS

Brazil

Pix is an FPS introduced by the Central Bank of Brazil in November 2020. Pix revolutionized the Brazilian payment landscape by providing a fast, secure, and accessible method for making electronic transactions.

As stated during interviews held with the Central Bank of Brazil, the role of the open-source approach in Pix is evident in certain aspects of the system. Although the Pix code itself is not open source, Pix relies on some open-source products. The system comprises three main systems developed internally by the Brazilian central bank (DICT, SPI, and ICOM) and includes a combination of open-source and proprietary market products.

The decision to use open-source or proprietary solutions is based on the maturity of available options for each technical need. For instance, if a specific technical challenge arises—for example, container orchestration—the central bank assesses the maturity level of open-source solutions such as Kubernetes. If a relevant production track record, an active community, and frequent updates exist, the bank adopts the open-source solution. Otherwise, it opts for proprietary software.

India

UPI (Unified Payments Interface) is an instant interbank electronic fund transfer system launched by the National Payments Corporation of India (NPCI) in 2016 to facilitate real-time money transfers between bank accounts using mobile phones, internet banking, or ATM services. Much like Pix, it enables instant payments between participants and is meant to be easily accessible for a wide set of users and interoperable with other payment methods and systems. The Immediate Payment System and the general Indian payment stack are generally not open source. However, the NPCI has released certain open-standard components, such as the BHIM service. The BHIM service, launched in 2016, was made open source in 2022 and is intended to allow banks to spin up a mobile banking app quickly, without spending excessive time and resources on development.[17]

ponents of their FPS on their own, with these solutions being entirely proprietary.

However, even in these proprietary systems, a reliance on open-source components is unavoidable: Relational database-management systems such as MySQL, caching systems for performance such as Varnish and Redis, durable messaging systems such Apache Kafka, and the Java language system and libraries are all significant open-source elements commonly used in financial systems.

This is also apparent in cloud services offered by the major service providers. Core facilities such as load balancers, data stores, virtual machines, container orchestration, and operating system software, for example, can be open-source components. Since they are running open-source software, so are their users.

New solutions have surfaced recently, notably in the realm of cloud computing, such as software-as-a-service. Companies such as Vocalink are offering fully hosted and managed FPS that, depending on the country context, may provide them with enhanced flexibility and a swifter time to market.[18] Nevertheless, the transaction-based pricing model employed by such services might lead to losing the benefit of lower per-transaction pricing, even though the per-transaction costs will come down as the utilization of the system increases.

5 RISKS AND BENEFITS OF OPEN-SOURCE TECHNOLOGIES

Institutions choosing whether to incorporate open-source components in their FPS must consider costs, risks, and benefits in relation to closed-source alternatives across the FPS life cycle. To support this analysis, the below section highlights issues that institutions must account for in each broad segment of the life cycle: design and conceptualization, implementation, and operation.

DESIGN AND CONCEPTUALIZATION

When choosing to design and develop an FPS, institutions must carefully consider their requirements, the capabilities of potential vendors and available technologies, costs, and internal resources and capabilities. In this context, deciding between open- and closed-source implementation carries the following key implications:

- **Requirement alignment:** As described in annex A, modern FPS have many functional requirements to deliver maximum value for participants and retail customers. Open-source projects may or may not have these features/functionalities; a key task in procurement is to ensure alignment between institutional requirements and the functionality of the solution under consideration. A well-considered solution (open source or closed source) should be able to address the most relevant needs of its users and be extensible for relevant use cases not directly addressed. In the longer term, the solution should be able to show missing features on its development road map. However, if the evaluated system is overly focused on certain use cases, and is unable or unwilling to implement user feedback, the immediate benefit of the system offering is limited. In addition, besides business and functional requirements, the system needs to meet certain security requirements in order to ensure the safe and sound functioning of the entire payment process.

  If open-source software and platforms exhibit limitations in their current and potential future functionalities, operators must rigorously evaluate the cost-effectiveness and efficiency of either outsourcing or internally developing these capabilities, and balance those costs against the commercial offerings. Furthermore, it is crucial for operators to thoroughly assess the compatibility of their and the participants' existing and future technological and operational infrastructure with open-source software and platforms. This analysis should consider potential conflicts with other components within a user's technology stack, as such incompatibilities can create challenges in seamlessly integrating and utilizing the open-source technology alongside other system elements. The compatibility of proprietary components with other system elements provided by others, whether open source or commercial, will depend on the openness of the proprietary systems to interoperability of data and function, and on their adherence to open standards (box 5).
- **Regulatory and legal implications:** When evaluating the feasibility of implementing an open-source FPS, a crucial consideration is the solution's ability to comply with regulatory requirements and pertinent standards. These requirements can vary significantly, encompassing

BOX 5 RELEVANCE OF OPEN STANDARDS FOR OPEN-SOURCE PROJECTS

Open standards are defined by a transparent development process that encourages broad participation, ensuring that the resulting specifications are accessible to everyone.[19] The authoritative source for these standards is the widely accepted formal technical specifications, which supersede any reference implementation. This approach facilitates the creation of interoperable software solutions by various organizations, allowing the solutions to operate harmoniously within a shared ecosystem.

Open standards and open-source projects are complementary. Open-source projects gain valuable insights and direction from open standards, particularly in terms of interface design and message formats for interoperability and portability, enhancing their ability to integrate and function seamlessly across different platforms. Open standards require implementations to confirm their suitability, establish a market presence, and gather feedback from implementors and users.

In the context of FPS, it is crucial for both proprietary and open-source projects to adhere to appropriate open standards when crafting their implementations. This adherence is essential to guaranteeing seamless interoperability and the smooth integration of FPS services into the broader payments and communications ecosystems. Common examples of these open standards, frequently embraced within FPS, encompass financial messaging standards, notably ISO 20022 and ISO 8583; both suites of international standards; and commercial industry standards of more narrow purpose, such as the EMVCo QR code specifications. And open industry standards for system hosting, API development, and inter-entity communication have been established and maintained for modern internet-based application ecosystems. These industry standards are essential to lowering the cost of development and participation by non-bank actors in market roles adjacent to an FPS.

both local and international regulations. For example, in a particular jurisdiction an FPS will need to comply with cybersecurity mandates that ensure that the system's security measures are robust enough to protect against threats and vulnerabilities. On an international level, an FPS will need to comply with established standards, such as the CPMI-IOSCO Principles for Financial Market Infrastructures.

Moreover, when developing an FPS using open-source technology, attention to intellectual property rights is essential. Open-source software is often perceived as freely accessible, yet it is regulated by diverse licensing agreements that can greatly influence both the functionality and legal standing of the system. A critical aspect to consider is the type of open-source license involved and the licenses under which dependent components are offered.

For instance, with a copyleft license, any distributed modifications or enhancements to the original software must be publicly disclosed using the same copyleft license. However, this requirement affects only cases where the modified software is distributed, not where it is used privately. The following explanation comes from the GNU General Public License (GPL) FAQ:[20]

> "The GPL does not require you to release your modified version, or any part of it. You are free to make modifications and use them privately, without ever releasing them. This applies to organizations (including companies), too; an organization can make a modified version and use it internally without ever releasing it outside the organization."

But this requirement can pose concerns when distributing the modified software is necessary to achieve payment system goals, especially when these changes pertain to operational details of the FPS that are confidential by regulation or contract.

By contrast, there are licenses that can be considered commercially friendly, as they are supportive of proprietary use without onward disclosure of enhancements. A careful legal analysis is required to assess the suitability of each license to each operator's context and needs. Therefore, a thorough understanding of open-source licenses and their implications for intellectual property rights is vital to ensuring the development of a secure, compliant, and effective FPS (box 6).

From a cyber risk perspective, it may help to incorporate the principle of security by design. It is typically much

BOX 6 TYPES OF OPEN-SOURCE LICENSES[21]

The GNU General Public License, version 3, is a widely adopted, free software license that ensures that end users—whether individuals, organizations, or companies—have the freedoms to run, study, share (copy), and modify the software. This strong copyleft license requires that the complete source code of licensed works and any modifications, including larger works that use a licensed work, be made available under the same license. It also mandates the preservation of copyright and license notices, and contributors must provide an express grant of patent rights.

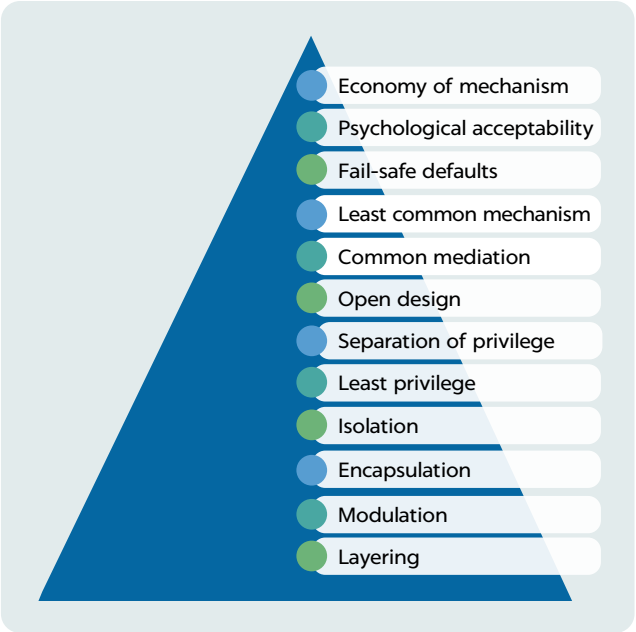
The Apache License 2.0, released by the Apache Software Foundation, is an open-source software license with permissive terms. It requires the preservation of copyright and license notices and includes an express grant of patent rights from contributors. Licensed works, modifications, and larger works can be distributed under different terms and without the need to provide source code.

The Mozilla Public License 2.0 is a simple copyleft license. This weak copyleft license requires that the source code of licensed files and their modifications be made available under the same license (or, in some cases, one of the GNU licenses). It also mandates the preservation of copyright and license notices, and contributors must provide an express grant of patent rights. However, larger works that incorporate the licensed work can be distributed under different terms, and the source code for files added in the larger work does not need to be provided.

The European Union Public License is a free software license initiated and approved by the European Commission. It aligns with the copyright laws of the European Union member states and is compatible with popular open-source software licenses, such as the GPL.

more effective to adopt secure design principles right from the beginning than to address cyber risk as an afterthought. While no techniques are foolproof, it is important (even vital) to develop relevant protection mechanisms that are based on different controls that need to be implemented to secure information systems. This includes payment systems. The U.S. National Centers of Academic Excellence in Cybersecurity list 13 baseline security principles that should be followed for developing relevant protection mechanisms.[22] Figure 1 shows the 12 design principles that are considered fundamental. Please refer to appendix C for further details on the secure design principles.

FIGURE 1 Secure Design Principles



IMPLEMENTATION

When implementing an FPS, institutions can modify several considerations by using an open-source or proprietary solutions. A key consideration in this decision-making process is the cost of implementation. This cost is significantly affected by two main elements: the adaptability of the chosen solution, and the capacity of the implementing institution to tailor the solution to meet its specific needs.

- **Adaptability:** Leveraging an open-source solution may allow for customizability. Since the source code is openly accessible, operators can modify independently and customize the software to suit their specific needs. However, this customization is not without its challenges. To alter the code effectively for specific use cases and functionalities, a high level of multidisciplinary expertise is necessary. The alteration and modification of code may inadvertently introduce vulnerabilities into the system, which may subsequently lead to an increase in cyber risk. Additionally, for successful customization, open-source projects must be accompanied by comprehensive documentation. This documentation is crucial for fully understanding the solution's scope and facilitating the implementation process. The challenge in customizing open-source solutions often lies in the complexity and scale of the software, which can make it difficult to modify without in-depth knowledge of its architecture and dependencies. Furthermore, integrating new modules and functionalities into an FPS can be a challenging and intricate task due to the dynamic nature of the payments industry. This sector is characterized by continual advancements, necessitating regular software updates and enhancements to keep pace with emerging trends and technologies. Consider the availability of open-market expertise in the open-source platform to augment operational staff.

  In fact, while open-source solutions offer adaptability and customizability due to accessible source code, entities implementing these solutions may still face challenges. Often, they need to engage a systems integrator to help customize the software, which can incur additional costs and potentially lead to the vendor lock-in issues they aimed to avoid with proprietary software.

In contrast, proprietary solutions may require that a vendor has the willingness and capability to tailor an implementation to the specific needs of the institution, which will also come at an additional cost. However, proprietary software vendors may be better equipped to provide targeted modifications and support.

- **Capacity:** As mentioned above, open-source components may be more directly configurable by the implementing institution than their proprietary counterparts, as this kind of customization requires substantial expertise and resources. Being a fully fledged member of an open-source community often requires having a strong community around the open-source project within a given institution, technical capabilities with respect to open-source development and management, and accommodative internal policies for engagement with open-source software and communities.

  Often, institutions seeking to develop FPS lack the sophisticated internal technical community necessary to take full advantage of the flexibility that open source offers. This applies particularly to the knowledge of different programming languages and development practices that may be necessary to develop and maintain open-source information systems, including payment systems. Secure coding practices form a vital component of developing secure payment systems. Instead, institutions can rely on external vendors to support the development of their payment system. Some open-source projects may have large vendor communities ready to support such an effort. Others are actively driving the growth of such communities. In scenarios such as these, using open source prevents vendor lock-in. However, it is also possible that few vendors exist for specific components of a hypothetical open-source implementation, causing costs to rise and timelines to widen.

  Moreover, to implement effective open-source FPS solutions, it is imperative that the implementing entity considers maintaining a dedicated team of system integrators and engineers, regardless of outsourcing the development in full or in part to external systems integrators. These professionals are essential for supporting the implementation process, ensuring that the open-source solutions are customized and optimized to meet the unique requirements of the institution. Their expertise not only facilitates the technical adaptation of these solutions but also ensures that the integration is seamless, secure, and in alignment with the institution's technological and operational framework. Additionally, these personnel will play a crucial role in promptly applying system updates to address arising issues. This applies to both open-source and proprietary solutions, albeit to varying extents based on the reliance on external vendors. But once an update becomes available, the operational staff will be depended on to schedule and deploy the updates appropriately as part of the ongoing management and operation of the FPS, unless this is part of the maintenance agreement.

OPERATION

Various benefits and costs of an open-source solution have an ongoing nature, persistent through the operation of an FPS.

- **Security:** Open-source software provides benefits with regards to security, particularly cybersecurity, but also comes with drawbacks. With the source code available

for inspection, users can verify the security and integrity of the software. Vulnerabilities may be discovered and addressed based on the collective efforts and expertise of the community.[23] Projects with effective governance structures often conduct their own internal security and license checks as well. But an important aspect to consider when evaluating the cybersecurity of open-source systems is that merely publishing the source code does not inherently ensure security. This also means that making the source code available does not guarantee it will be scrutinized for security flaws. Often, certain portions of the source code may not be reviewed promptly by security experts or other relevant evaluators due to resource constraints. Recent research from GitHub suggests that technical vulnerabilities in libraries that are widely used in open-source (and potentially other) information systems can remain undetected for an average of four years.[24] The use of unpatched programming libraries, or those that contain malicious code, represents a significant risk to the secure operation of information systems. This observation is particularly pertinent to FPS if the source code employed by these applications includes programming libraries that have not been thoroughly assessed for security risks.

Moreover, it should be noted that cybersecurity assessments are not a simple endeavor. The effectiveness of vulnerability assessments, penetration tests, and other cyber resilience techniques is contingent upon the resources at hand, and the available community resources may often be insufficient. For these assessments to be truly effective, they should ideally be conducted by professionals with specialized expertise, such as vulnerability assessors, penetration testers, and red team specialists.

It is also crucial to underscore the importance of having these assessments carried out by an independent party. This is particularly pertinent when the assessor is an internal member of the organization, as a lack of independence and potential conflicts of interest may deter the assessor from disclosing critical findings. Therefore, to ensure the integrity and thoroughness of the assessments, the independence of the assessor must be preserved.

Unsecure coding practices are an additional source of risk for open-source systems. While these practices may also affect and represent a challenge for proprietary applications, unsecure code is especially important in the case of open-source payment systems. Good programming includes the ability to produce secure code. If developers of payment systems lack the skills to code securely or fail to integrate secure coding practices into their workflow, the likelihood of creating vulnerable payment applications rises significantly.

Furthermore, given the need to depend on external developers to maintain an open-source project, it is possible for an open-source resource or component to become out-of-date or irrelevant to user needs. Open-source projects can be abandoned by their creators, which can also occur with proprietary software, where commercial vendors may cease to support a product line, albeit this risk is lower for operators. Moreover, proprietary software contracts require escrow of source code to mitigate such risk. In such cases, the source code is put in custody of a third-party escrow agent to protect the licensee. But in both cases, an operator is at risk, as significant resources and expertise are required to continue maintaining and updating the source-code packages.

When open-source projects are not actively updated, components can become increasingly stale and unable to evolve to meet the constantly changing needs of their userbase and new security threats; hence, implementation can become out-of-date/obsolete due to infrequent updates,[25] though this concern is uneven across open source as a whole. A report by Sonatype, a software supply-chain manager, noted only 11 percent of surveyed projects were being actively maintained.[26] Per this report, one in eight open-source software downloads had a known risk.

As users develop their own codebases, these implementations may become increasingly customized to specific circumstances, and their risks may become unknown to the community of developers, resulting in limited support or correction of code to ensure the security of the solution adequately. This concern can be mitigated in active open-source communities by sharing back enhancements: these enhancements are then incorporated into the managed software base of the open-source project, tested regularly, scanned for vulnerabilities, and packaged for release—all activities that then need not be performed by the adopting organization's staff. When adopting open-source systems, consider carefully the value, not just the cost, of sharing enhancements back to the community.

- **Maintenance and support:** Open-source projects may have active development communities that address user concerns on a project level, but no personnel will appear at a client site to conduct in-person troubleshooting, which may also be a risk for proprietary solutions lacking a maintenance agreement. As such, institutions must decide what level of support they are comfortable with, keeping their capacity constraints in mind and how and to what degree they will augment their own capacity with outside experts. Furthermore, as a project undergoes

more extensive customization, the ability of the community to offer maintenance and support may progressively diminish. An important factor is how well the FPS project shares enhancements back to the community and then uses the open-source project to host these enhancements such that they are maintained by the community, tested, packaged, and released. Fully custom software must be maintained by the developer that wrote it or by others that learn it well enough to do so, if the source is available to them.

For institutions opting for open-source FPS solutions, it becomes critical to acknowledge the necessity of a dedicated team of system integrators (internal or contracted). This specialized team is indispensable for proactive maintenance, precise troubleshooting, comprehensive security assessments, and the assurance of the system's consistent reliability. Unlike proprietary solutions, which often come with paid direct vendor support, open-source projects rely on the institution's capacity to manage and adapt the software to evolving needs and challenges, as well as the availability of open-market experts and system integrators. This team's role is not just to address immediate technical issues but also to ensure the long-term sustainability and effectiveness of the FPS solution within the organization's specific operational context.

- **System migration:** Entities considering the adoption of any solution must be aware of not only the initial benefits but also the potential long-term migration costs and risks associated with transitioning to different solutions in the future. These risks are inherently present in open-source projects, due to the possibility of diminished support from the community or the lack of a comprehensive open-source support system. But the risks are also present in, among other things, the face of secondary sanctions imposed on commercial providers, business dissolution, or commercial strategy pivots that cause the abandonment of a commercial offering. As mentioned, the escrow of proprietary software often mitigates part of this issue. If the software vendor goes out of business, fails to maintain the software, or breaches the contract, the licensee can obtain the source code to maintain and update the software independently. This can contribute to business continuity and reduces the risk of software becoming obsolete or unusable due to vendor issues. However, it is important to note that accessing the escrowed source code can also lead to additional costs for the licensee and that a deep understanding of the software, including its construction and packaging processes, is required to update or maintain the source code effectively.

Moreover, analyzing the total cost of ownership of any system should include potential migration costs. This analysis needs to account for the time and resources required to adapt to and integrate new systems, which can be substantial. However, additional risks warrant careful consideration, including compatibility issues and loss of customizations and enhancements. The outgoing solution may have been integrated with other systems or tailored to specific processes within an organization. These bespoke integrations can result in significant compatibility challenges during migration to new solutions.

COST IMPLICATIONS ACROSS THE LIFE CYCLE OF AN FPS

Choosing to adopt open source can have cost benefits for an institution, though it can lead to new costs as well. The primary cost savings afforded by an open-source solution is the lack of a licensing cost. Depending on the software being used, this savings can be significant. Further, some open-source projects have a full feature set, allowing institutions to use components "off the shelf" and without substantial further modification. However, given the technical demands of implementing and operating an open-source project, an institution can end up internalizing the significant development expense. This is particularly relevant in the case of costs related to the operation, maintenance, and upgrade of an open-source solution, particularly when the solution is not backed by an active and highly skilled community to support improvements and minimize threats. Balancing these internal costs, at internal labor rates, should be compared to the commercial rates for onboarding, training, updates, and other per-project costs of commercial systems.

It is important to note that when operators evaluate an open-source solution, they must consider costs that might not be immediately obvious at the outset but can become significant for the operator, participants of the FPS, and end users over time. However, it should be noted that a well-functioning FPS, regardless of its source base, requires constant tuning and adjustment to evolving circumstances. It is the fit-to-purpose of the system and its ongoing cost of support that should be studied carefully.

The hidden costs that could potentially be associated with all FPS projects also extend to integrating new use cases, functionalities, and adopting evolving standards. In the dynamic realm of the payments industry, where new use cases and standards are continually emerging, incorporating these advancements into existing systems is a complex and significant task. In open-source systems, this integration may

require extensive enhancements that may compel operators to hire contract assistance or to undertake development efforts in-house themselves and may lead to incremental costs on the operator's side. This aspect of platform evolution to meet the emerging needs of an FPS operator is particularly critical in the context of FPS functioning as a digital public infrastructure (DPI).[27] As a DPI, an FPS is expected to play a pivotal and cross-sectoral role, leading the charge in assimilating innovations into the broader digital ecosystem, which increases the pressure on the operators to anticipate and integrate innovations into an FPS.

As for cost-benefit analysis, no widely acknowledged framework applies to open-source projects. However, a concept that can aid cost-benefit analysis of open-source technologies is calculating the total cost of ownership (TCO). In the context of software, TCO is an estimate of an organization's overall expected spend to purchase, configure, install, use, monitor, maintain, optimize, and retire a technological component. In the context of open source, such a calculation would involve assessing not just the initial acquisition cost, which is often low or zero, compared to proprietary software, but also the long-term expenses associated with its deployment, maintenance, and operation. This includes costs for legal check, compliance management, support, training, integration with existing systems, potential customization, updates, and security measures, and the cost to financial institutions of onboarding the technology to the FPS and maintaining its use.

However, open-source software also presents unique cost dynamics and potential benefits that are not as easily quantified and that go beyond the TCO calculation. For a 2023 study on the economic impact of open-source software, the Linux Foundation surveyed 431 executives, including CEOs and CTOs from Fortune 500 companies, providing a granular look at how organizations perceive the value derived from open-source adoption.[28]

Survey respondents acknowledged multiple components when evaluating the benefits of open-source software, including its impact on TCO. Their responses are summarized in table 1.

In the context of FPS, the Mojaloop Foundation has collaborated with Glenbrook Partners to create an estimator designed to assess and compare the fundamental business propositions of various FPS platform configurations. This estimator evaluates the financial and operational nuances of four implementation approaches: solutions acquired through vendor licenses, those developed by in-house teams, and two variations of the Mojaloop platform. The first Mojaloop variation integrates Mojaloop's open-source software capabilities supplemented with components licensed from vendors for non-open-source requirements. The second variation also capitalizes on Mojaloop's strengths but opts for custom-built, non-open-source components crafted by internal development teams. This estimator seeks to elucidate two critical aspects: the comparative financial implications—encompassing development, implementation, and

TABLE 1 **Linux foundation—Survey responses**

MEDIUM TO NO BENEFIT	HIGH TO VERY HIGH BENEFIT
• Attractiveness of IT work environment: 50.72% reported medium to no benefit.	• Attractiveness of IT work environment: 49.29% reported high to very high benefit.
• Active community for knowledge exchange: 41.15% reported medium to no benefit.	• Active community for knowledge exchange: 58.85% reported high to very high benefit.
• Faster development speed: 34.45% reported medium to no benefit.	• Faster development speed: 65.55% reported high to very high benefit.
• High security of software: 66.83% reported medium to no benefit.	• High security of software: 33.17% reported high to very high benefit.
• High stability, low error susceptibility in open-source software code: 64.43% reported medium to no benefit.	• High stability, low error susceptibility in open-source software code: 35.57% reported high to very high benefit.
• Cost savings (lower TCO): 33.5% reported medium to no benefit.	• Cost savings (lower TCO): 66.51% reported high to very high benefit.
• Additional revenue opportunities/access to new markets: 73.21% reported medium to no benefit.	• Additional revenue opportunities/access to new markets: 26.79% reported high to very high benefit.
• Independence from proprietary providers: 45.45% reported medium to no benefit.	• Independence from proprietary providers: 54.54% reported high to very high benefit.
• Open standards and interoperability: 36.84% reported medium to no benefit.	• Open standards and interoperability: 63.16% reported high to very high benefit.
• Strong support from providers of open-source software: 73.68% reported medium to no benefit.	• Strong support from providers of open-source software: 26.32% reported high to very high benefit.

ongoing operational costs—of each FPS platform approach, and the time-to-market efficiency inherent in each implementation choice.

MATURITY OF AN ORGANIZATION TO LEVERAGE OPEN SOURCE

Implementing an FPS with open-source technologies demands robust operational and technological capabilities alongside vigilant management of the potential risks associated with these deployments. While a universally accepted model to gauge an entity's readiness for open-source adoption has yet to be established, there are initiatives aimed at creating frameworks to evaluate an organization's capability to manage and contribute to open-source projects effectively.

The TODO Group, a subfoundation of the Linux Foundation, introduced the concept of the open source program office (OSPO) to provide guidance on how organizations can formulate and implement their open-source strategies. An OSPO serves as a central point within an organization, responsible for establishing policies for using open-source and third-party components, ensuring license compliance, addressing legal considerations, and promoting community involvement and contributions. Furthermore, the TODO Group has outlined the five-stage OSPO Maturity Model, which describes an organization's journey from informal open-source use to integrating open-source strategies into core organizational decision-making processes. This model progresses from ad hoc adoption, through compliance and education, community engagement, and project leadership, to a phase in which the OSPO plays a crucial role in shaping the organization's technological direction.[29]

Similarly, the Open-Source Maturity Model crafted by FINOS and Wipro is specifically designed for the banking and financial sectors. It provides a framework for these institutions to evaluate their current open-source usage and develop strategies to enhance their open-source engagement. The model identifies five levels of maturity, starting from ad hoc, informal open-source usage to a stage in which open-source management is deeply embedded in the strategic and operational fabric of the organization, highlighting a commitment to ongoing improvement, active community participation, and leadership in the open-source ecosystem.[30]

6 LESSONS LEARNED

Open source in payments is still a growing space.

There is increasing interest from operators in the benefits that open source can bring to payments, including FPS. Mojaloop is one example that operators have just begun to assess actively. However, FPS do leverage open-source components, especially when it comes to external software and components such as cloud architecture, as the Central Bank of Brazil has shown with Pix.

Consider the following key factors before leveraging open source for FPS.

Before deciding to leverage open source to implement an FPS, institutions must carefully evaluate several factors that affect the entire life cycle of the system, from design and conceptualization to operation and long-term sustainability. The decision to use open-source or proprietary solutions involves balancing benefits such as flexibility and cost savings against challenges such as security risks, operational capacity, and regulatory compliance. The following factors are key aspects that institutions need to address when deciding how best to integrate open-source technologies within their FPS:

- **Design and conceptualization**
  - ***Requirement alignment:*** Ensuring that open-source solutions meet the functional, business, and security needs of the FPS is crucial. Institutions must confirm that these solutions align with current and future requirements, and open-source projects should adhere to open standards, such as ISO 20022, for seamless interoperability.
  - ***Regulatory and legal implications:*** Open-source solutions must comply with local and international regulations. Institutions must also carefully navigate intellectual property issues and open-source licensing to avoid unwanted disclosure of proprietary enhancements.
  - ***Security by design:*** Incorporating secure design principles from the beginning of the project is essential to mitigate cyber risks.
- **Implementation**
  - ***Adaptability:*** Open-source solutions offer customization but require significant expertise for them to be tailored to institutional needs, which can introduce security vulnerabilities if not handled properly. Proper documentation is key to managing the complexity of the implementation.
  - ***Capacity:*** Institutions need strong internal technical capabilities or vendor support to implement open-source FPS effectively. This includes building a dedicated team of system integrators to ensure seamless customization and security updates.
  - ***Customization versus vendor lock-in:*** Open-source solutions reduce vendor lock-in, but a lack of vendors for specific components may increase costs and lengthen timelines.
  - ***Secure integration:*** Operators should treat open source with the same rigor as proprietary code by selecting pre-vetted components, conducting initial and ongoing vulnerability assessments using soft-

ware composition analysis tools, and performing regular scans during the development and build stages.

- **Operation**
  - ***Security:*** While open-source solutions allow for code review and collective security checks, vulnerabilities can still remain undetected for years. Regular vulnerability assessments and third-party audits are essential.
  - ***Maintenance and support:*** Institutions must ensure that they have the internal capacity to maintain and troubleshoot open-source solutions, as they do not come with the same level of vendor support as proprietary solutions.
  - ***Strength and expertise of the open-source community:*** A highly active and skilled community ensures ongoing support, timely updates, and rapid responses to security issues. However, if the community lacks sufficient expertise or is inactive, slower updates, unresolved problems, and increased risks for the FPS may be the result.
  - ***System migration:*** Open-source solutions must consider long-term migration costs, including compatibility issues and the potential loss of customizations when transitioning to new systems.
  - ***Monitoring:*** It is essential to stay updated on open-source components and monitor for vulnerabilities, promptly updating or replacing unsupported components. Additionally, implementing a software bill of materials enhances transparency, documents component origins, and aids in managing vulnerabilities, reflecting a commitment to secure software-development practices throughout the software life cycle.
- **Maturity of an organization to leverage open source**
  - ***Operational readiness:*** Implementing open-source FPS requires robust operational and technological capabilities, and institutions need to manage potential risks. This requires a full assessment of the capacity of internal as well as available external resources, such as system integrators, to ensure seamless integration, ongoing support, and effective risk mitigation.
  - ***Maturity models:*** Tools such as maturity models can help operators assess and develop their ability to manage and contribute to open-source projects, enhancing strategic and operational alignment with open-source practices.
- **Cost implications across the life cycle of an FPS**
  - ***Total cost of ownership:*** Open-source solutions can reduce initial costs by eliminating licensing fees but may increase internal development and operational expenses. A full TCO analysis, including future maintenance, security, and migration costs, is necessary.
  - ***Hidden costs:*** The costs of integrating new functionalities, adhering to evolving standards, and adapting to new use cases must be considered. FPS as a DPI increases the pressure for continual innovation and resilience.

Choosing open-source technologies requires embracing a long-term vision and clearly assessing the capacity of the organization to adopt them.

It is crucial for any institution implementing an FPS to ensure that its long-term vision and goals for future enhancements align with the open-source solutions it chooses to adopt. And while this is also true for commercial solutions, this strategic alignment is a key factor in successful FPS development and should not be ignored when considering open-source solutions. This alignment plays a crucial role in the successful development of the FPS. When making decisions, institutions should appropriately weight the long-term adaptability and scalability of the solution, rather than overweight its immediate simplicity or ease of use.

It is important to consider how open-source solutions might affect or restrict future development possibilities. The chosen technology should not only meet current needs but also be adaptable to future requirements and enhancements. Adopting a long-term vision will help to avoid situations where initial convenience leads to later roadblocks in development, necessitating costly and time-consuming modifications or even complete system overhauls. The European Commission provides an example of taking a holistic approach toward assessing the adoption of open source (box 7).

Operators and regulators must assess the potential risks of open-source technologies not only for the FPS but for the broader payments ecosystem.

Like other technological options, open-source technology in FPS carries risks, in particular cyber and operational risks. A key factor to assess is whether open-source software can expose vulnerabilities to a broader audience, including potential attackers. These risks should be weighed against the capacity of the community to contribute to security updates and fixes.

Another risk involves the integration and compatibility with existing technological infrastructure. Open-source solutions might not always integrate seamlessly with other com-

BOX 7 EUROPEAN COMMISSIONS' OPEN SOURCE SOFTWARE STRATEGY[31]

The European Commission's Open Source Software Strategy 2020–2023 outlines the organization's commitment to leveraging open-source principles to enhance digital transformation across Europe. The strategy outlines the following principles regarding the adoption of open source:

- Open-source solutions will be preferred when equivalent in functionalities, total cost, and cybersecurity.
- We harness the working principles of open source; we innovate and cocreate, share and reuse, and together build user-centric, data-driven public services.
- We share our code and enable incidental contributions to related open-source projects.
- We strive to be an active member of the diverse open-source ecosystem.
- We make sure the code we use and the code we share is free from vulnerabilities by applying continuous security testing.
- We promote open standards and specifications that are implemented and distributed in open source.

ponents within an institution's technology stack, leading to additional challenges and costs in implementation.

Moreover, the evolving nature of open-source projects, with the possibility of them becoming outdated or abandoned, can pose operational risks, necessitating continuous monitoring and potential contingency planning.

Considering the critical role of FPS, including as DPIs, it is relevant for regulators to adopt a proactive approach toward the review of technologies underlying FPS. As mentioned above, if the operation of the FPS is critical for the overall functioning of the economy, the FPS may need to be included in national critical infrastructure. Additionally, the participation of FPS operators and service providers in cyber risk information–sharing platforms and public-private partnerships may prove to be vitally important. This is especially true when the exchange of information is needed to prepare for potentially adverse cyber events and to foster timely incident response and recovery from cyber incidents. The aforementioned process is essential in identifying and effectively mitigating the outlined risks.

Another relevant aspect of risk refers to the adoption of proprietary components with open-source peripherals. It is possible that vulnerabilities that remain undetected within proprietary components may become exposed and exploited in unexpected ways through a combination with open source. In this case, critical systems that are proprietary, which themselves may have undetected vulnerabilities, can be further exposed by threats affecting an open-source component used for developing, operating, or maintaining a proprietary software. A recent example illustrating these risks is the Log4Shell vulnerability discovered in Apache Log4j (box 8).

BOX 8 APACHE'S LOG4J INCIDENT

Apache Log4j is a widely utilized open-source logging framework supported by the Apache Software Foundation. Log4j is integral in data-logging processes across various applications and enterprise software systems, including both open-source and commercial systems. This critical vulnerability, when exploited, permitted attackers to inject malware, thereby gaining the ability to manipulate core elements of the targeted software and extract sensitive information. This undetected vulnerability had been present since 2013 and came to light and was subsequently addressed only in December 2021.[32]

At the same time, there can be reverse cases in which stable, secure, open-source software systems can be affected by security vulnerabilities in the underlying commercial operating platform, with vulnerabilities in commercial systems also passing undiscovered and unmitigated. The key learning is that the full software system and its operating environment must be considered, not just the individual components.

Another critical element is the implementation of a proactive strategy for monitoring existing risks in software supply chains. Following the example of the Log4j vulnerability, as of September 2023 a quarter of Log4j downloads were still of its vulnerable version. The Cyber Safety Review Board's

report *Review of the December 2021 Log4j Event*[33] underscores the critical role of open-source software in the digital ecosystem, given its widespread integration across numerous software components. However, it also highlights a significant challenge: the teams behind Log4j and the Apache Software Foundation, like many open-source projects, lack comprehensive oversight of the usage and application of their software. This means that the lack of control and oversight of open-source components must be addressed by a comprehensive analysis of such components before integration, as well as by a proactive strategy for monitoring evolving vulnerabilities and the overall landscape in which the component is deployed. Moreover, the board's report highlights challenges in maintaining open-source projects such as Log4j, which often lack dedicated security resources throughout the software-development life cycle.

Many open-source projects do not have coordinated teams for vulnerability disclosure and response to investigate and resolve reported issues. Furthermore, open-source projects usually develop code in a public manner, making security changes visible before official patches are released. This requires concurrent advisories and community responses to reduce the risk of malicious exploitation during the interim period (box 9).[34]

In sum, several steps can be taken to reduce cyber risk that stems from open-source FPS and other critical IT components used by the financial sector. Organizations that develop or utilize open-source information systems should employ secure design principles in the creation and implementation of payment systems. The source code used for these systems must be thoroughly evaluated and assessed for security issues. In addition to source-code reviews, conducting vulnerability scans, comprehensive penetration testing, and information system audits can be crucial for assessing the cybersecurity of FPS. At a minimum, an information system audit should be conducted by an independent assessor and encompass areas such as business continuity, resilience of critical business processes, access risks, and data accuracy, including the integrity of payment and transaction-related data.

BOX 9 VULNERABILITIES FOUND ACROSS OPEN-SOURCE CODEBASES

The *2024 Open-Source Security and Risk Analysis Report*[35] reveals that open-source components are prevalent in 96 percent of 1,067 codebases examined. A significant 84 percent of these codebases contain vulnerabilities; 74 percent face high-risk vulnerabilities, and 91 percent of the codebases use components that are outdated by 10 or more versions. In the financial and fintech sectors, the reliance on open source is high: 99 percent of codebases incorporate open-source elements, and 73 percent of these contain high-risk vulnerabilities. To enhance security, the report recommends the following measures:

- Integrate open-source components into the secure build process, treating them with the same rigor as proprietary code. This involves selecting pre-vetted components from an internal repository, conducting initial vulnerability assessments using software composition analysis tools, and performing ongoing scans during the development and build stages.
- Keep abreast of updates to open-source components and monitor for vulnerabilities. Upon detection of a vulnerability, it is crucial to evaluate the software to determine the extent of the component's usage and update it promptly. If the component is no longer supported, it is advisable to consider alternatives.
- Implement a software bill of materials. This is a detailed record that outlines the components and their supply-chain relationships within the software. It enhances transparency, documents the origin of components, and is instrumental in managing vulnerabilities. The presence of a software bill of materials may also reflect a supplier's commitment to secure software-development practices throughout the software life cycle.

7 CONCLUSIONS

Open-source technologies provide a distinct competitive advantage to FPS operators, compared to proprietary solutions. The most obvious benefit is the reduced cost of licensing. Additionally, open-source solutions offer greater flexibility and, hence, can be a viable mechanism for insourced projects and benefit from community support, especially when the community is active and thriving. However, leveraging open-source solutions for FPS is a complex decision that needs strategic consideration and alignment. Institutions must weigh the pros and cons of integrating open-source components in their FPS across the entire life cycle, including procurement, implementation, and operation.

Across the FPS life cycle, cost implications can be a major factor. Open-source solutions can save licensing costs, but calculations need to account for additional costs, including internal development and operational expenses, as well as costs associated with maintaining security, managing compliance, minimizing legal risks, integrating new functionalities, and keeping pace with evolving standards.

Institutions implementing FPS must align their decision with a long-term, forward-looking vision, ensuring that the chosen technology, be it open source or proprietary, not only satisfies medium-term requirements but is also capable of evolving to meet future demands. This is vital to ensure that the technology not only meets current needs but is adaptable to future enhancements, thus avoiding potential development roadblocks. A thorough cost-benefit analysis is essential, as open-source technologies, while offering cost savings and customizability, come with their own set of challenges. These include the need for significant in-house expertise, security risks, regulatory compliance, and intellectual property considerations. Another critical aspect of this decision-making process involves a thorough evaluation of the expertise, capacity, and level of support existing within the FPS operator and provided by the community backing open-source projects. This assessment is complex. The dynamics and capabilities of open-source communities vary significantly, as these depend on a more collaborative contribution and support, which can lead to varying degrees of responsiveness and expertise availability.

Indistinct of the technology, to ensure the robust and resilient operation of an FPS, it is crucial to anticipate and mitigate future risks associated with the software used to run it. This involves assessing the system's and the operator's operational and technical capacity against various challenges, including scalability, flexibility, and a range of risks—operational, cybersecurity, regulatory, and market-related. Addressing these scenarios will necessitate augmenting institutional capacity, and the operator must develop a tailored solution that combines resource allocation, strategic partnerships, and employee training programs.

Finally, the increasing critical role of FPS, including as DPIs, requires a proactive approach from regulators in reviewing their underlying technologies to mitigate risks effectively.

8 ACKNOWLEDGMENTS

Organization	Contributor
World Bank	Guillermo Galicia Rabadan (primary)
	Amar Kakirde (primary)
	Harish Natarajan
	Holti Banka
	Nilima Ramteke
	Thomas Piveteau
	Andrea Monteleone
	Kiyotaka Tanaka

The authors would like to thank Ghislain de Salins, Goran Vranic, Hunt La Cascia (all World Bank), for their valuable comments during the peer reviewing process.

APPENDIX A

SPECIFIC COMPONENTS OF AN FPS

An FPS accounts for the following technical and functional components:

- **Management, clearing, and settlement:** An FPS integrates diverse clearing and settlement models that are adaptable to various schedules and participation models across a range of use cases. It can also integrate liquidity-management tools, including but not limited to the establishment and monitoring of liquidity caps and limits.
- **Core transactions and use cases:** An FPS should be capable of handling different types of transactions, including instant credit and debit transfers, requests to pay, recalls/returns of funds, account verifications, investigations, proxies, and requests for information. Moreover, the system should be capable of being deployed across different use cases, including person-to-person transactions, government payments, merchant payments, and bulk payments.
- **Participation:** An FPS can address diverse participation models, including direct participants, indirect participants, transaction initiators, proxy-lookup participants, and third-party service providers.
- **Risk management:** An FPS is capable of integrating risk-management policies and enables monitoring and responding to a variety of risks, including financial risks, such as credit and liquidity risk. Moreover, it can integrate a fraud-detection and -prevention component, balancing rapid processing for immediate fund availability and detailed scrutiny to minimize fraud risk throughout the payment life cycle.
- **Operational and cyber resilience:** An FPS should be capable of meeting minimum cybersecurity standards and should include business-continuity and disaster-recovery plans. High availability is key, with robust recovery mechanisms that prevent issues such as data duplication or loss.
- **Scalability:** The system should handle significant increases in participants, transaction volumes, and diverse use cases efficiently.
- **Interoperability:** An FPS must be designed for seamless interoperability among payment service providers, facilitating communication and functionality across various payment products. It must be adaptable to current and future use cases in the payment ecosystem.
- **Access channels:** An FPS must facilitate access to core functionalities through various channels, including USSD, SMS, internet and mobile banking solutions, wallets, NFC, and QR codes.

APPENDIX B

COMPONENTS OF AN OPEN-SOURCE SOLUTION AND ECOSYSTEM

IMPLEMENTING AND MAINTAINING AN OPEN-SOURCE PROJECT

Implementing an open-source project, given its inherently do-it-yourself nature, requires significant effort and coordination. While the literature on adopting open-source software in the context of a fast payment organization is limited, the broader question of implementing open source is well explored.

Engaging in an open-source project can start from two places: building internally or joining an existing project. In some cases, it may make sense for an organization to start a completely new open-source project. If a given need—in this case, fast payments—is not addressed in the broader market, an organization may choose to build internally. Such an organization can then invite other organizations to join the project and contribute to the community. This is a resource-intensive path and can add complexity to the already complex process of developing a project internally—that is, instead of simply managing organization-specific stakeholders, the project-governance model would need to address the concerns of other community members. Of course, this approach has benefits, since community members can provide expertise and resources to assist in the development of a given project.

Joining an existing project is easier and faster than starting one but comes with its own challenges. Given that the organization is joining an existing community, it needs to evaluate that community on several characteristics, including but not limited to the following:

- **Leadership and governance**
  - An institution choosing to join an existing community should become comfortable with the governance structure of the open-source community it is joining. If the institution intends to be a major contributor to the project, it should ensure that its views are represented in the governance structure.
- **Relevance of underlying software**
  - The project the institution is joining should be relevant to the needs of the institution.
- **Mission and vision**
  - A joining institution should ensure that the mission and vision of the open-source project do not conflict with its own mission and vision.
- **Value alignment**
  - A joining institution should ensure that its values are in line with those of the open-source project community.
- **Communication channels and style**
  - Depending on the need and desire for communication, a joining institution should ensure that the open-source project is able to meet those needs.
- **Maturity**
  - A joining institution should make sure that the open-source organization is appropriately mature to handle the needs of the institution.

- **Licensing models**
  - The licensing model should be suitable to the use cases of the joining institution, and the open-source project should actively manage the inbound licenses of dependencies used in the project for compatibility with its outbound license. Some projects have copyleft provisions that require all development arising from the initial open-source project also to be open source, which can be challenging for institutions.

The depth of this diligence can vary, depending on the level of involvement the organization wants to have in the project. However, given the benefits of collaboration that are offered through open-source projects, it makes sense to leverage the benefits of the project's open-source community. Once this diligence is completed, the organization can then proceed to join the community and adopt any community requirements that may exist.

Along with this process, an organization may need to adjust internal resources and policies to enable implementation and participation in the community. Such adjustments may include internal governance changes, establishing a point of contact, establishing dedicated development teams, implementing training on the use/interaction with the new project, and more. Identifying a rapporteur and key point of contact for the open-source community may assist the flow of communication between in-house and community stakeholders.

A key decision that the organization will need to make is how to implement new project releases. This entails several things. Bearing in mind the project's own release schedule, the organization will need to develop a road map for its own use. Often, open-source projects will not deliver exactly what a given organization needs in its design, planned or otherwise. As a result, organizations will need to insert the development of its own features into such an organization-specific road map. Furthermore, while organizations can leverage testing done by the open-source project, if it exists, it often makes sense to conduct this internally as well. And the prestaging of community-released updates in a preproduction environment enables the safe observation of the new releases before opening to production traffic.

All of this is resource intensive, compared to leveraging a closed-source solution. The size of costs and their timing may be quite different in these two approaches and require a business analysis.

To summarize the institutional structures and considerations that must be addressed, organizations should take the following steps:

- Establish a governance system for its projects
- Ensure the appropriate level of expertise for development through training, hiring, or outsourcing
- Establish a road map for releases
- Determine the level of engagement with the open-source community
- Implement the infrastructural requirements for the project; in the context of payments, this includes data centers, servers, and hosting environments

APPENDIX C

SECURE DESIGN PRINCIPLES

The U.S. National Centers of Academic Excellence in Cybersecurity list the baseline security principles that should be followed for developing relevant protection mechanisms for information systems.[36] The principles apply to both open and proprietary information systems, including payment systems.

Economy of mechanism: Economy of mechanism refers to the fact that security measures, such as control mechanisms that are designed and implemented in both hardware and software, should be as simple and small as possible. Simple and small designs are usually easier to test and verify in detail. When the security design is complex or difficult to understand, there are more opportunities for an adversary to discover subtle weaknesses to exploit that may be difficult to identify in advance. In general, the more complex the security mechanism, the higher the probability (likelihood) that the mechanism may have security flaws and vulnerabilities. Again, simpler mechanisms are likely to have less exploitable weaknesses and require less maintenance. In addition, because configuration-management issues are simplified, updating or replacing a simple mechanism becomes a less intensive process. In practice, this is one of the most challenging principles to implement. In most organizations, there is a constant demand for new features in both hardware and software, which complicates the security-design task. The best that can be done is to keep this principle in mind during system design, to try to eliminate unnecessary complexity.

Fail-safe default: The fail-safe default implies that access decisions need to be based on permission, as opposed to exclusion. This means that the default situation should be a lack of access, and where the protection scheme of the organization identifies the conditions necessary for access to be granted. This approach is characterized by a better failure mode than its alternative, which grants access even in cases when something might go wrong. A design or implementation mistake in a mechanism that gives explicit permission tends to fail by refusing permission, a safe situation that can be quickly detected. On the other hand, a design or implementation mistake in a mechanism that explicitly excludes access tends to fail by allowing access, a failure that may long go unnoticed in normal use. For example, most file-access systems work on this principle, and all protected services on client/server systems work this way.

C**omplete mediation:** This component refers to the fact that every access must be checked against the access-control mechanism. Systems should not rely on access decisions retrieved from a cache. In a system designed to operate continuously, this principle requires that, if access decisions are remembered for future use, careful consideration should be given to how changes in authority are propagated into such local memories. File-access systems provide an example of a system that complies with this principle. However, typically, once a user has opened a file, no check is made to see if permissions change. To fully implement complete mediation, every time a user reads a field or record in a file, or a data item in a database, the system must exercise access control. This resource-intensive approach is rarely used.

Open design: The idea behind open design is that the

security mechanism should be open, instead of secret. For example, when looking at the way encryption may be implemented, while the encryption keys that are used to encrypt data need to remain secret, the encryption algorithm that is used should be open to scrutiny and review by outside experts.

Separation of privilege: Separation privilege is frequently defined as a practice that includes the use of several privilege attributes that are needed to achieve access to a restricted resource. Day-to-day operations are executed in a lower privileged-access regime. A prime example of separation of privilege includes two-factor authentication. Multifactor authentication requires several (at least two) authentication techniques, such as a password and biometrics, to authorize a user. Separation of privilege can also be used to refer to any task that is divided based on specific privileges. For example, administrative tasks may be restricted to a separate account, while everyday activities are conducted with low-privilege accounts.

Least privilege: The concept of least privilege is associated with the idea that every process or task and every user of a specific system should function with the least set of privileges that are necessary to conduct the task. Role-based access control is an access-control principle and method based on the concept of least privilege. Each role is assigned only the permissions that are needed to perform specific tasks.

Least common mechanism: The idea of the least common mechanism refers to the fact that the design should minimize the functions that are shared by different users, the end result of which is to provide mutual security. The principle helps reduce the number of unintended communication paths and reduces the amount of hardware and software on which all users depend, making it easier to verify if there are any undesirable security implications.

Psychological acceptability: One of the most important concepts of a secure design is psychological acceptability. This means that the control mechanisms that are implemented should not interfere excessively with the everyday operations of users and the organization itself. The mechanisms implemented should not hinder the organization's functioning. If the internal control mechanisms are deemed excessive and unusable by the staff, employees might choose to ignore the controls wherever this might be possible. As a result, the cyber risk associated with such practices might increase. It is also worth noting that the implemented security controls should make sense to the employees and fit the mental model of the users. In short, these mechanisms should not be too burdensome. In addition, if the control mechanisms do not make sense to the employees, the chance of making mistakes also increases.

Isolation: The concept of isolation has three parts. In general, public access systems should be isolated from the more sensitive or critical systems that contain sensitive data, processes, or other assets of the fast payment service operator or service provider. For some of the more sensitive information systems and assets, physical isolation of the critical systems may be considered. In other cases, a defense-in-depth approach, using logical security controls, may be implemented. The second aspect of isolation refers to the fact that processes and files of individual users should be isolated from one another except where access is specifically needed. All modern operating systems offer the capability to provide separate space for individual users, with relevant protection mechanisms for the prevention of unauthorized access. The third component of isolation deals with the need to isolate security mechanisms, such as internal controls, in such a way that prevents unauthorized access to the security controls.

Encapsulation: Encapsulation is typically a form (subset of isolation) that is founded on object-oriented functionality. Security is provided by encapsulating or enclosing a collection of procedures as well as data objects in a separate domain in such a way that the internal structure of a data object is accessible only to the procedures of the protected subsystem and the procedures may be called only at designated entry points.

Modularity: The use of modular architecture is one of the key components of secure design. This principle implies the use and adoption of a modular architecture and the development of security functions as separate, protected modules. For example, functions associated with the encryption of data and information should use common security modules or services. Security modules should be portable to newer technologies in an easy manner without too much (excessive) effort.

Layering: As noted in some of the previous guidance, operational risk stems from people, processes, systems, and external events. Security controls (that is, internal control mechanisms) should be developed to ensure a defense-in-depth approach. This applies to, among other things, payment system processes. Control mechanisms should be designed and implemented to ensure overlapping protection addressing the operational risk factors mentioned above. This means that if one control mechanism fails, there are other overlapping controls that will not leave the system unprotected.

ENDNOTES

1. https://www.hbs.edu/faculty/Pages/item.aspx?num=65230
2. Specifics of the technical components of an FPS are included in annex A.
3. Considerations and Lessons for the Development and Implementation of FPS, World Bank, 2021.
4. Ibid.
5. Encyclopedia by Kaspersky. n.d. "Closed-Source Software (Proprietary Software)", https://encyclopedia.kaspersky.com/glossary/closed-source/
6. Open-source licenses are categorized into two main types: copyleft and permissive. Copyleft licenses, such as the GNU General Public License (GPL), allow the modification or customization of software, but they require that any modifications also be distributed under the same license, ensuring that the software remains open source. Permissive licenses, such as the MIT License or Apache License, permit the use and modification of open-source software with fewer restrictions. Unlike copyleft licenses, they do not require the distribution of modified source code.
7. St. Laurent, Andrew M, *Understanding Open-Source and Free Software Licensing* (Sebastopol, CA: O'Reilly Media, 2008), 4.
8. FINOS. 2022. *The 2022 State of Open Source in Financial Services*, https://www.finos.org/hubfs/FINOS_Report_010323.pdf.
9. Microsoft Open Source. n.d. "Our Program," https://opensource.microsoft.com/program/.
10. Brewer, Eric, and Abhishek Arya. 2022. "Shared Success in Building a Safer Open Source Community." *The Keyword*, May 12, 2022, https://blog.google/technology/safety-security/shared-success-in-building-a-safer-open-source-community/#:~:text=Since%20under%2Dmaintained%2C%20critical%20open,of%20critical%20open%20source%20projects.
11. Zimmerman, Evan J. 2024. "Privatized Open Source." *CMR Insights*, June 24, 2020, https://cmr.berkeley.edu/2020/06/privatized-open-source-software/
12. Strange, Angela. 2021. "Open Source Is Finally Coming to Financial Services." Andreesen Horowitz, October 15, 2021, https://a16z.com/open-source-is-finally-coming-to-financial-services/.
13. In the context of payment systems, open-loop systems refer to payment networks that allow transactions to be processed across multiple institutions, enabling users to use their payment instruments and transaction accounts at a wide range of merchants and access points domestically or internationally.
14. Interledger: https://interledger.org/case-studies/making-digital-payments-affordable-and-simple-for-everyone-everywhere/{~?~URL
15. Bandura, Romina, and Ramanujam, Sundar R. 2021. *Developing Inclusive Digital Payment Systems*. Center for Strategic and International Studies, https://www.jstor.org/stable/pdf/resrep35090.pdf.
16. Helms, John. 2023. "10 Open-Source Software Security Risks." *ConnectWise*, September 27, 2023, https://www.connectwise.com/blog/cybersecurity/open-source-software-risks.
17. Mint. 2022. "NPCI Announces BHIM App Open-Source License Model." *Mint*, November 9, 2022, https://www.livemint.com/industry/banking/npci-announces-bhim-app-open-source-license-model-11668001560218.html.
18. https://b2b.mastercard.com/news-and-insights/success-story/modernizing-latin-american-payments/
19. https://www.itu.int/en/ITU-T/ipr/Pages/open.aspx
20. https://www.gnu.org/licenses/gpl-faq.html#GPLRequireSourcePostedPublic
21. https://documents1.worldbank.org/curated/en/672901582561140400/pdf/Open-Source-for-Global-Public-Goods.pdf
22. Stallings, W., and L. Brown. 2014. *Computer Security: Principles and Practice*, 3rd ed. Upper Saddle River, NJ: Pearson Prentice Hall.
23. Google Open Source. n.d. "Why Open Source?," https://opensource.google/documentation/reference/why.
24. https://www.cpomagazine.com/cyber-security/open-source-vulnerabilities-take-four-years-to-spot-says-github/
25. PYMNTS. 2017. "When Open Source Opens the Door for Cybersecurity Risks." *PYMNTS*, April 21, 2017, https://www.pymnts.com/news/b2b-payments/2017/black-duck-open-source-security-cybersecurity-license-software-enterprise-app/.
26. Krill, Paul. 2023. "Report Finds Few Open Source Projects Actively Maintained." *InfoWorld*, October 12, 2023, https://www.infoworld.com/article/3708630/report-finds-few-open-source-projects-actively-maintained.html.
27. DPIs, generally understood as interoperable, open, and inclusive systems supported by technology to provide essential, societywide, public and private services, can play a critical role in accelerating this transformation in an inclusive way.
28. https://project.linuxfoundation.org/hubfs/LF%20Research/Measuring%20the%20Economic%20Value%20of%20Open%20Source%20-%20Report.pdf?hsLang=en
29. https://todogroup.org/resources/guides/
30. https://github.com/finos-labs/osmm
31. https://commission.europa.eu/about-european-commission/departments-and-executive-agencies/digital-services/open-source-software-strategy_en

32. National Cyber Security Centre, n.d., "Log4j Vulnerability—What Everyone Needs to Know," https://www.ncsc.gov.uk/information/log4j-vulnerability-what-everyone-needs-to-know; and CISA, 2021, "Mitigating Log4Shell and Other Log4j-Related Vulnerabilities," https://www.cisa.gov/news-events/cybersecurity-advisories/aa21-356a.
33. https://www.cisa.gov/sites/default/files/publications/CSRB-Report-on-Log4-July-11-2022_508.pdf
34. https://www.cisa.gov/sites/default/files/publications/CSRB-Report-on-Log4-July-11-2022_508.pdf
35. https://www.synopsys.com/software-integrity/engage/ossra/ossra-report
36. Stallings, W., and L. Brown. 2014. *Computer Security: Principles and Practice*, 3rd ed. Upper Saddle River, NJ: Pearson Prentice Hall.


WORLD BANK GROUP