The Journal of Systems and Software 159 (2020) 110452

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Maintaining interoperability in open source software: A case study of the Apache PDFBox project

Simon Butler[a,*], Jonas Gamalielsson[a,*], Björn Lundell[a,*], Christoffer Brax[b], Anders Mattsson[c], Tomas Gustavsson[d], Jonas Feist[e], Erik Lönroth[f]

[a] University of Skövde, Skövde, Sweden
[b] Combitech AB, Linköping, Sweden
[c] Husqvarna AB, Huskvarna, Sweden
[d] PrimeKey Solutions AB, Stockholm, Sweden
[e] RedBridge AB, Stockholm, Sweden
[f] Scania IT AB, Södertälje, Sweden

ARTICLE INFO

Article history:
Received 28 June 2019
Revised 10 October 2019
Accepted 21 October 2019
Available online 22 October 2019

Keywords:
Standards
Software implementation
Software interoperability
Community open source software
Portable document format

ABSTRACT

Software interoperability is commonly achieved through the implementation of standards for communication protocols or data representation formats. Standards documents are often complex, difficult to interpret, and may contain errors and inconsistencies, which can lead to differing interpretations and implementations that inhibit interoperability. Through a case study of two years of activity in the Apache PDFBox project we examine day-to-day decisions made concerning implementation of the PDF specifications and standards in a community open source software (OSS) project. Thematic analysis is used to identify semantic themes describing the context of observed decisions concerning interoperability. Fundamental decision types are identified including emulation of the behaviour of dominant implementations and the extent to which to implement the PDF standards. Many factors influencing the decisions are related to the sustainability of the project itself, while other influences result from decisions made by external actors, including the developers of dependencies of PDFBox. This article contributes a fine grained perspective of decision-making about software interoperability by contributors to a community OSS project. The study identifies how decisions made support the continuing technical relevance of the software, and factors that motivate and constrain project activity.

1. Introduction

Many software projects seek to implement one or more standards to support interoperability with other software. For example, interconnected systems implement standardised communications protocols, such as the open systems interconnect stack, and web standards, including the hypertext transfer protocol (HTTP) and the secure sockets layer (SSL), to support information exchange and commercial activities (Wilson, 1998; Treese, 1999; Ko et al., 2011).

As businesses and civil society — governments at national and local level, and the legal system — move away from paper documents (Lundell, 2011; Rossi et al., 2008) to rely increasingly on digitised systems, the implementation of both communication protocols and document standards becomes ever more crucial (Rossi et al., 2008; Wilson et al., 2017; Lehtonen et al., 2018). Standards are written by humans, and despite the care taken in their creation they are imperfect, vague, ambiguous and open to interpretation when implemented in software (Allman, 2011; Egyedi, 2007). Furthermore, practice evolves so that implementations, often seen as the *de facto* reference for a standard, can diverge from the published standard as has been the case with the JPEG image format (Richter and Clark, 2018). Indeed, practice can, for example with HTML, CSS and JavaScript, repeatedly deviate from standards, sometimes with the intention of locking users in to specific products (W3C, 2019a; Bouvier, 1995; Phillips, 1998) and with the consequence that web content becomes challenging to implement and access (Phillips, 1998), and to archive (Kelly et al., 2014).

While software interoperability relies on standards, different software implementations of a given standard are interpretations

* Corresponding authors.
E-mail addresses: simon.butler@his.se (S. Butler), jonas.gamalielsson@his.se (J. Gamalielsson), bjorn.lundell@his.se (B. Lundell), christoffer.brax@combitech.se (C. Brax), anders.mattsson@husqvarnagroup.com (A. Mattsson), tomas.gustavsson@primekey.com (T. Gustavsson), jonas.feist@redbridge.se (J. Feist), erik.lonroth@scania.com (E. Lönroth).

https://doi.org/10.1016/j.jss.2019.110452

of the standard that may not be fully interoperable (Egyedi, 2007). Consequently, the developers of software implementations will become involved in a discourse to find a common understanding of the standard that supports interoperability, as illustrated by Allman (2011), Lehtonen et al. (2018), and Watteyne et al. (2016). The means by which interoperability is achieved varies. The Internet Engineering Task Force (IETF) (IETF, 2019a), for example, uses a process, often summarised as "Rough consensus and running code" (Davies and Hoffmann, 2004), that requires interoperability between independent implementations is achieved early in the standardisation process (Wilson, 1998). An increasing proportion of software that implements communication and data standards, particularly where it is non-differentiating, is developed through collaboration by companies working in community open source software (OSS) projects (Lundell et al., 2017; Butler et al., 2019). By *community OSS project* we mean OSS projects managed by foundations or are collectively organised (Riehle, 2011), where many of the developers are directed by companies and other organisations, and collaborate to create high quality software (Fitzgerald, 2006). Examples of this process include OSS projects under the umbrella of the Eclipse Internet of Things Working Group (Eclipse IoT Working Group, 2019), and LibreOffice (The Document Foundation, 2019). In many cases and domains both OSS and proprietary solutions are available for the same standard and need to interoperate to remain relevant products. While the literature documents the process of standardisation, and the technical challenges of implementing standards compliant software, there is little research that focuses on how participants in OSS projects decide how to implement a standard, and how to revise their implementation to correct or improve its behaviour. To explicate the challenges facing community OSS projects developing standards compliant software and the day-to-day decisions made by contributors this study investigates the following research question:

> *How does a community OSS project maintain software interoperability?*

We address the research question through a case study (Gerring, 2017; Walsham, 2006) of two years of contributions to the Apache PDFBox®[1] OSS project. The PDFBox project is governed by the Apache Software Foundation (ASF) (ASF, 2019a) and develops and maintains a mature (Black Duck, 2019) Java library and tools to create and process Portable Document Format (PDF) documents (Lehmkühler, 2010). PDFBox is used in other OSS projects (Apache Tika, 2019; CEF Digital, 2019; Khudairi, 2017), and as a component in proprietary products and services. PDFBox is described further in Section 3.2.

Developed in the 1990s, PDF is a widely-used file format for distributing documents, which are created, processed and read by many different applications on multiple platforms. Versions of PDF are defined in a number of specifications and standards documents, including formal (ISO) standards, that implementers need to follow to ensure the interoperability of their software. There is evidence that the PDF standards are challenging to implement (Bogk and Schöpl, 2014; Endignoux et al., 2016), that the quality of PDF documents varies (Lehtonen et al., 2018; Lindlar et al., 2017), and that the dominance of Adobe's software products creates user expectations that need to be met by the developers of other PDF software (Gamalielsson and Lundell, 2013; Endignoux et al., 2016; Amiouny, 2017; 2016). In the following section we provide a background description of PDF, and also review the related academic literature.

[1] PDFBox is a registered trademark of the Apache Software Foundation.

Section 3 details reasons for the purposeful sampling (Patton, 2015) of PDFBox as the case study subject. We also identify the data sources investigated for the case study and give an account of the application of thematic analysis (Braun and Clarke, 2006) to identify semantic themes in the types of decisions concerning the interoperability of PDFBox made by contributors to the project and the factors influencing those decisions.

Through the analysis of the data we identified four fundamental types of decision made concerning the interoperability of PDFBox related to compliance with published PDF specifications and standards. The types of decision and the technical circumstances in which they are made are described in Section 4. We also provide an account of the factors identified that influence those decisions including resources, knowledge, and the influence of external actors, such as the developers of other PDF software, and the creators of documents. We discuss the challenges faced by the PDFBox project in Section 5 including the technical challenges faced by the developers of PDF software, and potential solutions. Thereafter, we consider how the behaviour of contributors to the PDFBox project sustains the project in the long term. Lastly, we present the conclusions in Section 6 and identify the contributions made by this study.

2. Background and related work

2.1. Standards development and interoperability

The development of standards for information and communications technologies is undertaken by companies and other organisations using a range of approaches, e.g. whether the technology is implemented before the standard is developed, and the working practices of the standards body involved. One perspective is that standards have two different types of origin. Some standards are specified by standards bodies, e.g. ISO and ITU. While others arise through extensive or widespread use of a particular technology, regardless of whether it was developed by one company or collaboratively (Treese, 1999). Another perspective is that standards are either *requirement-led* or *implementation-led* (Phipps, 2019). Phipps, a director (and sometime President) of the Open Source Initiative, argues the primary use of the requirement-led model is where standardisation is used to create a market, for example the development of 5G (Nikolich et al., 2017). In contrast, implementation-led standards are developed to support an innovation in software or data format that has been adopted by a wider audience than the creating company and standardisation is necessary to support interoperability. A third view is provided by Lundell and Gamalielsson (2017) who identify standards that are developed before software, software that is implemented and then forms the basis of a standardisation process (including that of PDF), and the development of standards in parallel with software. The latter process is identified as being of increasing importance in the telecommunications industry (Wright and Druta, 2014), and examples can be found in the standardisation process for internet protocols managed by the IETF (IETF, 2019a). The IETF emphasises interoperability at an early stage of protocol development, rather than technical perfection (Bradner, 1996; Wilson, 1998; Bradner, 1999). The process of developing interoperability between low powered devices in the IoT domain is described by Ko et al. (2011). They record the development of the internet protocol (IP) in 6LoWPAN to provide interoperable communications stacks for two IoT operating systems Contiki-OS and TinyOS. The interoperable implementations are then used to determine whether the solutions achieved are practicable for the types of IoT devices expected to use them (Ko et al., 2011).

A further approach to interoperability is the development of implementations of standards, particularly communication protocols,

Table 1
Selected PDF versions and ISO standards.

Version	ISO Standard	Year	Comment
PDF v1.0		1993	First published PDF specification.
PDF v1.4		2001	Improved encryption, added XML metadata, and pre-defined CMaps.
PDF v1.5		2003	Added JPEG 2000 images and improved encryption.
PDF/A-1	ISO 19005-1:2005	2005	An archive format for standalone PDF documents based on PDF v1.4.
PDF v1.7		2006	Extended range of support for encryption.
	ISO 32000-1:2008	2008	ISO standardised version of PDF based on Adobe's PDF v1.7 specification.
PDF/A-2	ISO 19005-2:2011	2011	An archive format for standalone PDF documents based on ISO 32000-1:2008.
PDF/A-3	ISO 19005-3:2012	2012	An extension of PDF/A-2 to support file embedding.
PDF v2.0	ISO 32000-2:2017	2017	Revision of ISO 32000-1:2008.

in OSS projects. Companies participating in the Eclipse IoT Working Group (2019), for example, collaborate, sometimes with competitors, in OSS projects to develop implementations of open communications standards used in the IoT domain that then support their products (Butler et al., 2019). Examples include the implementation of the Open Mobile Alliance's (OMA, 2019) lightweight machine to machine (LWM2M) protocol in Leshan (Eclipse Foundation, 2019b) and Wakaama (Eclipse Foundation, 2019c), and the constrained application protocol (CoAP) (Shelby et al., 2014) in Californium (Eclipse Foundation, 2019a). Additionally, the collaborative OSS project serves to identify and document cogent misinterpretations and misunderstandings of the standard (Butler et al., 2019).

2.2. PDF standards and interoperability

Adobe Systems developed PDF as a platform-independent, interchange format for documents that can preserve presentation independently of the application and operating system. In 1993, the first PDF specification was made freely available and a number of revisions of the specification have been published since (see Table 1). Some versions of the specification have been published as ISO standards (e.g. ISO 32000-1:2008 and ISO 32000-2:2017), including specialised subsets of the PDF format for the print industry (e.g. ISO 15929:2002 and ISO 15930-1:2001), and engineering applications (e.g. ISO 24517-1:2008).

PDF documents vary in size and complexity from single page tickets, receipts and order summaries, through academic papers, to very large documents, such as Government reports, and books. Consequently, PDF documents may have short lifespans, or have a significantly longer life as business and legal records, particularly as organisations move away from paper. Many different software packages exist to create, display, edit and process PDF files. Further, a significant problem for long-term use of PDF is that many documents will outlive the software used to create them (Gamalielsson and Lundell, 2013), so will require standards compliant software that can faithfully reproduce the documents to be available at some arbitrary point in the future.

PDF software, therefore, does not work in isolation; it *must* interoperate with other software to the extent that implementations need to be able to process documents created by other software, regardless of how long ago, and to create documents that other implementations can read. Furthermore there is the requirement that those documents be readable many years in the future, particularly in the case of documents such as contracts and official documentation issued by governmental agencies. These requirements are not a theoretical exercise, they are practical requirements that already pose problems for organisations and businesses. For example, in the dataset examined for this article there is evidence that contractors for the Government of the Netherlands have created many thousands of official academic transcripts as PDF documents that do not comply with the PDF specifications and are, at best, problematic to process (see mailing list thread Users-1, Table 5 on p 8).

PDF is a complex file format that is used to create documents with a rich variety of content including text, images, internal document links, indexes, fillable forms, and digital signatures. Each version of the PDF standard cites normative references — other standards — that form part of the standard and are described as "... indispensable for the application of this document" in ISO 32000-1:2008 (ISO, 2008). The normative references include standards for fonts, image formats, and character encodings. In addition, several normatively referenced standards include normative references themselves (and so on). For example, among the normative references of ISO 32000-2:2017 is part 1 of an early revision of the JPEG 2000 ISO standard (ISO/IEC 15444-1:2004) which in turn has 13 normative references, including 10 IEC, ISO and ISO/IEC standards. The specifications and standards also define the declarative programming language that describes PDF documents, as well as the expected behaviours and capabilities of programs that create and process PDF documents. The size and complexity of the PDF specifications and ISO standards themselves pose a daunting challenge for software developers implementing them. The recently published ISO 32000-2:2017 standard, for example, consists of 984 pages and has 90 normative references (ISO, 2017). Further challenges complicate the development of software that works with PDF files. A key challenge is the common perception that the Adobe Reader family of software applications are the *de facto* reference implementations of the PDF specifications and standards to which the performance of other implementations is compared (Amiouny, 2016; Lehtonen et al., 2018). Another source of difficulty is the *Robustness Principle* (Allman, 2011), otherwise known as *Postel's Law*, which is applied in Adobe's Reader products, and stated by Postel, in the context of communication protocols, as, "... be conservative in what you do, be liberal in what you accept from others." (Postel, 1981). In practice, PDF reading and processing software implements repair mechanisms to allow malformed files to be read, within limitations. The limitations, however, are only documented in the behaviour of Adobe's products.

2.3. Related work

A key aspect of software interoperability is the agreement and documentation of data formats and communication protocols in specifications and standards. There are many practical challenges to the standardisation process, and a number of approaches have been tried. Ahlgren et al. (2016) argue that open standardisation processes are needed to support interoperability in the IoT domain. An example can be found in the development of implementations of the 6TiSCH communications protocol for low-power devices (Watteyne et al., 2016). Watteyne et al. describe an iterative process of interoperability testing between implementations and how the lessons learnt through testing inform further iterations of the standardisation process. Another example is the standardisation of the QUIC protocol. Originally implemented by Google, QUIC has been in use for some 6 years and a standard is being developed by an IETF committee (IETF, 2019b; 2019c; Piraux et al., 2018).

Piraux et al. (2018) evaluated the interoperability of fifteen implementations of QUIC finding some shortcomings in all. The tests developed by Piraux et al have since been incorporated in the test suites of some of the implementations tested (Piraux et al., 2018).

Standardisation processes can take a long time, and consequently may be seen by some as an inhibitor of innovation. De Coninck et al. (2019), for example, cite the slowness of the QUIC standardisation process as motivation for a proposed plugin mechanism to extend QUIC. They have proposed, implemented and investigated a flexible approach where applications communicating with QUIC negotiate which extensions to QUIC to use during connection set-up (De Coninck et al., 2019).

Standards are also long-lived, and require review and revision in response to developments in both practice and technology. The Joint Photographic Expert Group (JPEG) have initiated a number of standardisation efforts to update the 25 year old JPEG standards for image files, including the JPEG XT project (JPEG, 2019). Richter and Clark (2018) identify how JPEG implementations differ from the standard, and the difficulties of applying the JPEG conformance testing protocol published in ISO 10918-5:2013 (ISO, 2013) to current implementations. Richter *et al.* identify two key issues. Firstly, the evolution of a body of practice building on the standard during the 25 years since it was made available, which motivates the standardisation review. Secondly, parts of the current standard are not used in practice, and may no longer need to be part of any revised standard (Richter and Clark, 2018).

The standardisation of HTML and CSS, and other web technologies followed a different path. Standards for both HTML and CSS have been developed by the World Wide Web Consortium (W3C) (W3C, 2019b) since the 1990s (W3C, 2019a), initially under the auspices of the IETF (Bouvier, 1995). During the browser wars (Bouvier, 1995) companies would add functionality to their browsers to extend the standard, and encourage web site developers to create content specifically for innovative features found in one browser. The process of developing websites to support variations in HTML became so onerous for developers that practitioners campaigned for Microsoft and Netscape to adhere to W3C standards (Phillips, 1998; WaSP, 2019).

Previous research on the development of PDF software in two OSS projects found developers adopted specific strategies to support interoperability (Gamalielsson and Lundell, 2013). Specifically, developers would exceed the specification, and mimic a dominant implementation so that their software complied with that implementation. In addition, the study illuminated difficulties developers had interpreting the PDF standard. One issue identified was the lack of detail in parts of the specification that made software implementation imprecise, and unreliable. Another concern expressed was that the complexity of the specification inhibited implementation (Gamalielsson and Lundell, 2013). Indeed, analyses of PDF from the perspective of creating parsers have found the task to be challenging (Bogk and Schöpl, 2014; Endignoux et al., 2016). As part of their investigation of PDF, Endignoux et al. (2016) identify ambiguities in the file structures that were used to discover bugs in a number of PDF readers. Bogk and Schöpl (2014) describe the experience of trying to create a formally verified parser for PDF. They advise that the creators of future file format definitions should ensure that the format is "... complete, unambiguous and doesn't allow unparseable constructions." (Bogk and Schöpl, 2014) In practice, the complexity of PDF specifications can lead to significant security vulnerabilities in software implementations (Mladenov et al., 2018a; 2018b).

The PDF/A standards (see Table 1) are used in document preservation. An area of concern is the management of documents that do not comply with the PDF/A standards. Lehtonen et al. (2018) identify the complexity of the problems faced by those handling documents, and explore mechanisms through which documents might be repaired so that they are "well-formed and valid PDF/A files." The team behind the development of veraPDF, a PDF/A validator, identify difficulties interpreting the PDF/A standard (Wilson et al., 2017) to be able to create validation tests representing a clear understanding of the standards. Additionally, Wilson et al. (2017) record the need to limit the scope of the validation tests implemented in veraPDF because of the scale of the task, particularly in the validation of normative references such as JPEG 2000. Lindlar et al. (2017) record the development of a test set of PDF documents to test the conformance of PDF files with the structural and syntactic requirements of ISO 32000-1:2008. The authors argue that a test set used to examine basic well-formedness requirements is helpful in digital preservation, as it simplifies the detection of specific problems as a precursor application of document repair techniques (Lindlar et al., 2017).

In summary, previous research shows the necessity of standardisation for software interoperability, and details approaches to standardisation. Research has also identified how practice can deviate from standards, and in the case of PDF the practical difficulties of developing software, and the challenges of creating mechanisms to evaluate standards compliance. The challenges of implementing standards have also been recorded. However, there is a lack of research that examines the nature of day-to-day practical decision-making of software developers when implementing a standard.

3. Research approach

We undertake a case study (Gerring, 2017; Walsham, 2006) of a single, purposefully-sampled (Patton, 2015) community OSS project that focuses on the challenges contributors face when creating and maintaining interoperable software and how they collaborate to resolve problems.

3.1. Case selection

Apache PDFBox was selected as a relevant subject for the case study for several reasons. Firstly, for PDFBox to have any value for users it must be able to interoperate with other software that reads and writes PDF documents. As such, it must implement sufficient of the PDF specifications and standards to be perceived as a viable solution. Secondly, the PDF specifications and standards are complex and documented as challenging to implement, with the additional requirement that implementations need to process a wide variety of conforming and non-conforming documents to emulate the functionality of a dominant implementation. Thirdly, though the software produced by the OSS project is most likely to be used in a business setting, PDFBox is an ASF project and is independent of direct company control. Consequently, contributors to PDFBox are obliged to rely on cooperation with others in the community to achieve their goals. Fourthly, the PDFBox project actively develops and maintains software, responds to reports of issues with the software, and releases revisions of the software frequently.

The scope of the investigation is the publicly documented work contributing to nine releases of Apache PDFBox between the release of v2.0.3 in September 2016 and the release of v2.0.12 in October 2018. The period investigated was specifically chosen to include the publication of the ISO 32000-2:2017 standard, also known as PDF v2.0, in August 2017.

3.2. Case description

The Apache PDFBox project develops a Java library and command line tools that can create and process PDF files. The library is relatively low level and can be used to create and process PDF documents conforming to different versions of the PDF specifications

Table 2
Data sources used in the case study.

Data Source	Location
Commits Mailing List	http://mail-archives.apache.org/mod_mbox/pdfbox-commits/
Developers Mailing List	http://mail-archives.apache.org/mod_mbox/pdfbox-dev/
Users Mailing List	http://mail-archives.apache.org/mod_mbox/pdfbox-users/
GitHub Mirror	https://github.com/apache/pdfbox
Jira Issue Tracker	https://issues.apache.org/jira/projects/PDFBOX/

and ISO standards (see Table 1 for examples). In development since 2002, and an ASF governed project since 2008, PDFBox is maintained by a small group of core developers and an active community of contributors. PDFBox is a dependency of some other ASF projects, including Apache Tika (Apache Tika, 2019), and other OSS projects, including the European Union funded Digital Signature Services project (CEF Digital, 2019). PDFBox is used to parse documents in one version of the veraPDF validator (veraPDF, 2019), as well as being used in proprietary software products and services. PDFBox was also part of the software suite used by journalists to extract information from PDF files amongst the documents collectively known as the Panama Papers (Khudairi, 2017; ICIJ, 2019).

At the time of the study, the most recent major revision of PDFBox, v2.0.0, had been released in March 2016 and maintenance releases have generally been made approximately every two to three months since. In addition, the project maintains an older version, v1.8, in which bugs are fixed, and releases made less often. The overwhelming majority of bug fixes for the 1.8.x series are backported from the 2.0.x series. The project is also working towards a major revision in v3.0.

3.3. Data collection

The core data for the case study consists of the online archives of activity in the PDFBox project. Using the PDFBox website (Apache PDFBox, 2019) we identified the communication channels available for making contributions, and the resources available for users of the software and contributors (see Table 2). Three public communication channels can be used to make contributions: the Jira issue tracker, and *developers* and *users* mailing lists. In addition there is a *commits* mailing list that reports the commits made to the PDFBox source code repository through messages generated by the version control system. A read-only mirror of the PDFBox source code is also provided on GitHub.

Mailing list archives identified were downloaded from the ASF mail archives (ASF, 2019b) and the GrimoireLab Perceval component (Bitergia, 2019) was used to parse the Mbox format files and convert them into JSON format files. The JSON files were then processed using Python scripts to reconstruct the email threads and write the threads out in emacs org-mode files for analysis (org-mode[2] is a plain text format for emacs that supports text folding and annotation). The Jira issue tracker tickets were retrieved in JSON format using the Jira REST API (Atlassian, 2019). The JSON records for each ticket were then aggregated and processed by Python scripts to create org-mode files containing the problem description and the comments on the ticket.

3.4. Data analysis

The data gathered from the PDFBox project was analysed using the *thematic analysis* framework (Braun and Clarke, 2006).

Initially, the first author worked systematically through all the collected data to identify the email threads and issue tracker tickets that address the topic of interoperability in any regard. The mailing list threads and issue tracker tickets cover a wide range of topics including project administration as well as help requests, and potential bug reports. Key factors considered included reference to the capabilities of PDFBox in comparison to other PDF processing software and mention of any PDF specification or standard or any of its normative references, such as font and image formats. During this phase, email threads were reconstructed where parts of conversations with the same subject line had been recorded in the archives as separate threads.[3]

The set of candidate email threads and issue tracker tickets were then examined in more detail to identify discussions in which decisions were made concerning the implementation of functionality related to the PDF specifications and standards, and their normative references in PDFBox and other software. Mailing list threads and issue tracker tickets where no clear decision was articulated were ignored for analytical purposes, as were discussions where it was judged there was insufficient information given for any decisions made to be clearly understood.

The conversations recorded in mailing list threads and issue tracker tickets contain the technical opinions and judgements of domain experts, including the core developers, and often contain explicit reference to PDF specifications and standards. Where there was no specific reference to a standard in a conversation, the topic of the discussion was used to determine relevance through comparison with other conversations on the topic explicitly linked to the PDF standards by contributors. At the end of the process, 111 mailing list threads and 394 issue tracker tickets had been identified for further analysis. Coding was also used at this stage to annotate the discussions, and particularly the decisions made, to help identify the nature of the problems being addressed, the relationship between the problems and the PDF standards and other PDF software, and the outcome of the decision-making process.

The corpus of 505 mailing list and issue tracker discussions was then analysed in depth by the first author to identify candidate semantic themes to describe the types of decision being made, and to identify candidate thematic factors influencing the decisions made. The coding from the previous phase supported the grouping of decision types and the development of semantic themes. Additional coding undertaken at this stage was used to identify factors influencing decisions and to develop a set of candidate thematic factors.

In the subsequent phase, all authors discussed the candidate decision types and factors alongside illustrative discussions taken from the corpus. A set of four semantic themes and seven thematic factors was agreed, and their consistency with the larger body of evidence reviewed by the first author.

4. Findings

This section describes the semantic themes identified through thematic analysis that categorise the decisions made by contributors to PDFBox regarding maintenance of its interoperability. Each decision type is illustrated with examples. Thereafter we provide

[2] https://orgmode.org/.

[3] Each email header contains a reference to the message it replies to. Sometimes the reference can be omitted when replying to a mailing list message.

Table 3
Types of software development decisions related to the PDF specifications and standards in the Apache PDFBox project.

Decision Type	Description
Improve to match de facto reference implementation	A decision taken in the context of improving or correcting PDFBox to match the de facto reference implementation.
Degrade to match de facto reference implementation	A decision taken in the context of degrading the compliance of PDFBox with a PDF specification or standard so that the behaviour matches that of an Adobe implementation.
Improve to match standard	A decision taken in the context of improving or correcting the behaviour of PDFBox to meet a PDF specification or standard.
Scope of implementation	A decision taken about the extent of the PDFBox implementation.

Table 4
Apache PDFBox Jira issue tracker tickets referenced in Section 4.1.

Decision type	Issue tracker ticket
Improve to match de facto reference implementation	PDFBOX-3513
	PDFBOX-3589
	PDFBOX-3654
	PDFBOX-3687
	PDFBOX-3738
	PDFBOX-3745
	PDFBOX-3752
	PDFBOX-3781
	PDFBOX-3789
	PDFBOX-3874
	PDFBOX-3875
	PDFBOX-3913
	PDFBOX-3946
	PDFBOX-3958
Degrade to match de facto reference implementation	PDFBOX-3929
	PDFBOX-3983
Improve to Match Standard	PDFBOX-3914
	PDFBOX-3920
	PDFBOX-3992
	PDFBOX-4276
Scope of implementation	PDFBOX-3293
	PDFBOX-4045
	PDFBOX-4189

an account of the main factors that motivate and constrain the outcomes of the types of decision made.

4.1. Decision types

We identified four major types of decision related to the implementation of the PDF specification and standards in the PDFBox project (see Table 3), each of which is described below with illustrative examples. We also provide descriptions of the thematic factors identified that, in combination, influence the decisions made.

4.1.1. Improve to match de facto reference implementation

Much of the work of PDFBox contributors is focused on trying to match the behaviour of Adobe's PDF software. The PDFBox core developers and many contributors treat the Adobe PDF readers as *de facto* reference implementations of the PDF specifications and standards (e.g. PDFBOX-3738 [4] and PDFBOX-3745 — PDFBox Jira issue tracker tickets referred to in Section 4.1 are listed in Table 4), and use the maxim that PDFBox should be able to process any document the Adobe PDF readers can. As one core developer explains:

> "There is the PDF spec and there are real world PDFs. Not all real world PDFs are correct with regards to the spec. Acrobat, PDFBox and many other libraries try to do their best to provide workarounds for that. We typically try to match Acrobat ..." (PDFBOX-3687).

The ISO 32000-2:2017 standard (ISO, 2017, pp. 18-19) identifies two classifications of PDF processing software: PDF readers and PDF writers. Accordingly, developers trying to match the Adobe implementations face two major challenges. The first is to be able to process the same input that Adobe software does. The second is to create output of similar quality to that produced by Adobe software. There are also two types of output of PDF software: the document created, and how given document is rendered on screen or in print. To "try to match Acrobat" (PDFBOX-3687), documents created by PDFBox should, insofar as is possible match those output by Adobe software so that they are rendered consistently by other software, and the expectation is that PDFBox, and software created using it, should also render documents with similar quality to the Adobe implementations (e.g. PDFBOX-3589 & PDFBOX-3752).

The convention in software that reads PDF files is to apply the Robustness Principle (Allman, 2011; Postel, 1981) so that documents that are not compliant with PDF specifications and standards can be processed and rendered, insofar as is possible (e.g. PDFBOX-3789). Exactly what incorrect and malformed content should, or can, be parsed into a working document is not specified by the PDF specifications and standards. The exemplar for developers is the behaviour of the Adobe Readers, as well as the behaviour of other PDF software.

PDF documents consist of four parts: a header, a body, a cross reference table, and a trailer. The header consists of the string "`%PDF-`" and a version number, followed, on a second line, by a minimum of four bytes with a value of 128 or greater so that any tool trying to determine what the file contains will treat it as binary data, and not text. The trailer consists of the string "`%%EOF`" on a separate line, immediately preceded by a number on one line representing the offset of the cross-reference table and the string "`startxref`" on the line before that (see Fig. 1). A PDF parser[5] reads the first line of a file and then searches for the "`%%EOF`" marker and works backwards to find the cross-reference table using the offset on the preceding line, and to read the trailer that confirms the number of objects referenced in the table, and the object reference of the root object of the document tree. The parser should then be able to read all the objects in the PDF file.

Where the cross-reference table is missing or damaged, PDF parsers *may*, according to the ISO 32000-1:2008 standard (ISO, 2008, p. 650), try to reconstruct the table by searching for objects in the file[6] (see Fig. 2). In practice, Adobe software appears to apply the Principle of Robustness more widely so that a wide range of problems, for example with fonts, are also tolerated by the parser.

[4] The PDFBox Jira issue tracker tickets referenced have URLs of the form `https//issues.apache.org/jira/browse/PDFBOX-NNNN` where 'NNNN' is the four digit number of the ticket. For example, PDFBOX-3738 has the URL https://issues.apache.org/jira/browse/PDFBOX-3738.

[5] There are also 'linearised' PDF files intended for network transmission where the trailer and cross-reference tables precede the body.

[6] The repair mechanism is why, sometimes, Adobe software applications offer the opportunity for the user to save a newly opened document.

```
xref
0 8
0000000000 65535 f
0000000015 00000 n
0000000078 00000 n
0000000135 00000 n
0000000254 00000 n
0000000352 00000 n
0000000385 00000 n
0000000416 00000 n
trailer
  <<
    /Root 1 0 R
     /ID [<AA28D985BB05C6D21792C4AF6718B2DF>
         <AA28D985BB05C6D21792C4AF6718B2DF>]
    /Size 8
  >>
startxref
518
%%EOF
```

Fig. 1. An example PDF file cross-reference table and trailer.

```
1 0 obj
  <<
    /Type /Catalog
    /Version /1.4
    /Pages 2 0 R
  >>
endobj
```

Fig. 2. An example PDF document catalogue object.

The work required to resolve issues of this nature varies in scope. Sometimes the source code revision is relatively trivial; a simple change to make the parser more lenient because the document author's intention is clear. For example, PDFBOX-3874 where a small change is made to a font parser so that it will accept field names in the font metadata that are capitalised differently to the specification. Similarly, in PDFBOX-3513, the PDFBox core developers identify an error in the ISO 32000-1:2008 standard as the underlying cause of an observed problem with PDFBox. One column of a table specifies two types (a name and a dictionary) for the value of an encoding dictionary for Type 3 fonts (ISO, 2008, p 259), the next column of the table clearly specifies that the field must be a dictionary. The contributor who encountered the document, proposes a revision to the parser to accommodate the error (PDFBOX-3513). One core developer comments that "... we've never encountered a file with the problem you've presented." Another core developer points out that there is no guidance in the specification on how to treat a Type 3 font that does not have an encoding dictionary. Instead of improvising a fallback encoding, the core developers argue that there may be a case to ignore the font specified in the document as it cannot be reliably used, and the parser is not revised given the rarity of the problem.

Adobe and other PDF software sometimes exceed the specifications and standards. In PDFBOX-3654, for example, a file is found that renders in many other applications, but not in PDFBox. The problem is a font that is encoded in a hexadecimal format, and the standard is unequivocal on the subject:

> "Although the encrypted portion of a standard Type 1 font may be in binary or ASCII hexadecimal format, PDF supports only the binary format." (ISO, 2017, p. 351)

The source code is revised to support the font encoding and the core developer processing the issue observes:

> "So the font is incorrectly stored. But obviously, Adobe supports both, so we should too." (PDFBOX-3654)

In some cases the Adobe software extends the specifications and standards through the implementation of additional functionality that reflects wider practice. Often the only documentation of the additional functionality is in the implementation, and other implementers only discover the change when differences in behaviour are reported to them. For example, a report in PDFBOX-3913 shows that Adobe software and PDF.js[7] process and render a Japanese URI, which PDFBox can not. The ISO 32000-2:2017 standard specifies that the targets of URIs (links) should be encoded in UTF-8. In both applications the URI is encoded in UTF-16, which is necessary to represent some Japanese characters used in domain names, but exceeds the standard. Revisions are made to PDFBox (documented in PDFBOX-3913, PDFBOX-3946, and PDFBOX-3958) to support UTF-16 for URIs and implement the same functionality as both Adobe and PDF.js.

PDFBox contributors also find instances where documents created by the software are not rendered as expected by Adobe's software. In these cases, typically, there is a difference in the model in documents created by PDFBox and the model that Adobe expects. In some cases a great deal of work is required to understand how Adobe and other readers interpret the PDF document. In PDFBOX-3738 work is undertaken to understand how the output of digitally signed files is interpreted by Adobe and other reader products. The acquired knowledge is then applied so that PDFBox can create documents that can be read and rendered with digital signature displayed by other PDF software. The developers also identify a related problem, documented in PDFBOX-3781, that affects documents with forms and digital signatures.

Merging PDF files can be a difficult problem for implementers to solve. PDFBOX-3875 records the challenges faced when merging two documents where the internal bookmarks are structured using slightly different representations in the document model. In the merged document some of the bookmarks do not work as expected. The initial assessment by one of the core developers is that the cause is within the PDFBox source code and is "... probably a bug. Not the kind that will be fixed quickly ...". One approach used by the core developers to evaluate how best to solve the problem is to merge the documents using other applications, including Adobe software, and to examine the document created following the merge. Work is started to try to create a viable solution by emulating the document resulting from merging the files using Adobe software, but further problems are encountered and the work is not completed.

4.1.2. Degrade to match de facto reference implementation

As noted already, developers of PDF software, including the PDFBox developers, tend to view Adobe PDF software implementations as a gold standard. However, Adobe's software developers do not always implement the PDF specifications and standards in the way that others might, and on occasions, implement solutions that can be seen as incorrect. Consequently, developers of PDF software then need to determine how they might degrade the adherence of their software to the PDF specifications and standards to match Adobe's implementations.

PDFBOX-3929 begins in a discussion on the PDFBox users mailing list where a user observes that PDF documents created by PDFBox with floating point numbers used for field widget border

[7] PDF.js is a widely used open source PDF reader implemented in JavaScript, see https://mozilla.github.io/pdf.js/.

Table 5
Apache PDFBox mailing list threads referenced.

Reference	Mailing list archive URL
Users-1	http://mail-archives.apache.org/mod_mbox/pdfbox-users/201804.mbox/⟨DB5PR01MB18629047633DB004EEFE111E85880@DB5PR01MB1862.eurprd01.prod.exchangelabs.com⟩
Users-2	http://mail-archives.apache.org/mod_mbox/pdfbox-users/201709.mbox/⟨CY1PR04MB226578FDD86270ED2F4A8358829 70@CY1PR04MB2265.namprd04.prod.outlook.com⟩
Users-3	http://mail-archives.apache.org/mod_mbox/pdfbox-users/201709.mbox/⟨CY1PR04MB2265E98C627098CDBA44DB2F82940@CY1PR04MB2265.namprd04.prod.outlook.com⟩
Users-4	http://mail-archives.apache.org/mod_mbox/pdfbox-users/201711.mbox/⟨CAKLHnLzfzvtUtxM-Kj2a1EbNa_YMG5qHnUy55PQeqoAV6KBLsQ@mail.gmail.com⟩
Users-5	http://mail-archives.apache.org/mod_mbox/pdfbox-users/201710.mbox/⟨3723506D-D663-4EB6-832F-AC052EDC230B@madlon-kay.com⟩

widths, are rendered by Adobe XI and Adobe DC without a border (Users-2 and Users-3 in Table 5). The borders of other annotation types are unaffected.

The width of borders drawn around annotations, such as form fields, are defined in PDF documents in two ways: a *border array* holding three or four values, or in some cases a *border style dictionary* (an associative array) that includes a value for the width of the border in points. In both cases the value to specify the width is defined as a *number*. PDF specifications and standards define two numeric types *integer objects* and *real objects*. The ISO 32000 standards then say "... the term *number* refers to an object whose type may be integer or real." ISO, 2008, p. 14; ISO, 2017, p. 24). ISO 32000-2:2017, for example, is explicit where fields are required to hold integer values, and uses the term number for other numeric fields.

Both versions of the ISO 32000 standard define the border array using the following sentence:

> "The array consists of three numbers defining the horizontal corner radius, the vertical corner radius, and border width, all in default user space units." (ISO, 2008, p. 384; ISO, 2017, p. 465)

Accordingly, the interpretation of the standards used in PDFBox agrees with the standard; border width can be specified with a floating point number. However, the Adobe reader software expects an integer, and ignores non-integer values, such as 3.0, by treating them as having a value of zero. Consequently, the PDFBox implementation was revised so that annotations in documents created by PDFBox will be rendered with borders by Adobe DC. A bug report was also made to Adobe support, saying that the standard had been interpreted incorrectly.

A closely related issue is found in a thread on the users mailing list (Users-4) where a developer reports that that the Adobe reader implementations behave in an unexpected way. This time the concern is the border drawn around a URI action annotation, or a link. The border is defined in the standard as described above, but the Adobe reader implementations interpret the values 1, 2, and 3 as meaning a thin, medium and thick border respectively. The PDFBox API documentation is updated to describe how the Adobe reader implementations interpret the border width value.

A contributor reports in PDFBOX-3983 that Acrobat Reader fails to display some outlines and borders where the *miter limit* is set to a value of zero or less. The miter limit indicates how junctions between lines should be drawn. The ISO 32000-1:2008 standard states:

> Parameters that are numeric values, such as the current colour, line width, and miter limit, shall be forced into valid range, if necessary. (ISO, 2008, p124)

The statement was revised in ISO 32000-2:2017 by the replacement of "forced" with "clipped" (ISO, 2017, p. 157).

Accordingly, one interpretation might be that a compliant PDF reader would be able to display a document correctly regardless of the value of the miter limit recorded because it would automatically correct the value. However, Adobe implementations appear not to correct the value. The user reporting the problem supplies a patch so that the miter limit in documents created by PDFBox will contain miter limit values that are positive, and the simple fix allows Adobe software to display the document. OpenPDFtoHTML, another OSS project, has also encountered the same problem and takes similar action.[8]

4.1.3. Improve to match standard

The PDFBox implementation is also revised to meet the requirements of the PDF standards and normative references, independently of the need to match the performance of Adobe products.

The use of multi-byte representations of characters in Unicode character encodings such as UTF-16 require some careful processing by PDF parsers because some single byte values can be misinterpreted. The single byte value 0x20 represents the space character in fonts encoded in one byte. In multi-byte character encodings the byte 0x20 may be part of a character and so should not be treated as a single byte. Two kinds of operator can be used in PDF documents to position text, one of which should be used with multi-byte font encodings so that single byte values that form part of multi-byte characters are not mis-interpreted. A patch is contributed in PDFBOX-3992 so that PDFBox fully supports the operator used to justify multi-byte encoded text to comply with the ISO 32000-1:2008 standard.

The PDF/A group of standards define an archive format for PDF. The demands of the standards are high, and compliance requires a great deal of attention to detail during document preparation. In general, the PDF/A standards constrain the types of content that can be present in compliant files, and sometimes make very precise demands on the quality of embedded resources. The veraPDF Project develops a freely available validator for PDF/A files. PDFBox also implements 'preflight' functionality to validate documents against the requirements of PDF/A-1b (the ISO 19005-1:2005 standard) and there are examples where the implementation is revised to match the performance of the veraPDF validator when differences are found. For example, a bug in the preflight validator is found in PDFBOX-4276 and the functionality corrected so that the incorrect output is now detected as veraPDF would. In PDFBOX-3920 a user reports that font subsets created by PDFBox do not include all the data required by the PDF/A-2 standard (ISO 19005-2:2011). The PDFBox source code is modified so that the output meets the standard.

The number of revisions to the PDF specifications and standards mean that occasionally it is found that PDFBox does not implement a particular feature or capture all the data in a PDF document. A contributor reports a problem with PDFBox where a field is ignored during parsing that leads to content being rendered that is supposed to be hidden. The user provides a patch in PDFBOX-3914

[8] https://github.com/danfickle/openhtmltopdf/issues/135.

which forms the basis of an update to the source code so that the field is imported and the document rendered correctly.

4.1.4. Scope of implementation

The core developers also make decisions about the scope of the software implemented by the PDFBox project. The question of what functionality forms the scope of the PDFBox implementation arises in some bug reports and feature requests, and has multiple dimensions.

PDFBox is not intended to be a comprehensive solution for creating, processing or rendering PDF documents. The project charter, or mission statement says:

> "The Apache PDFBox library is an open source Java tool for working with PDF documents. This project allows creation of new PDF documents, manipulation of existing documents and the ability to extract content from documents. Apache PDFBox also includes several command-line utilities. Apache PDFBox is published under the Apache License v2.0." (Apache PDFBox, 2019)

PDFBox relies on some external libraries to provide functionality, especially in the area of image processing. There is no need for the PDFBox project to reimplement the wheel, particularly in technically demanding domains. A further difficulty is that image processing provision within the core Java libraries is incomplete, and varies between Java versions. Some functionality, such as the JPEG 2000 codec, is no longer maintained and is difficult for OSS implementers to adopt because of the licence used and potential patent issues (discussed further in Section 4.2.6). Java provision for image processing is changing and, with Java v9, functionality is gradually being returned to the core libraries. However, the JPEG 2000 codec remains outside the main Java libraries. Further, PDFBox core developers often recommend the use of the Twelve Monkeys plugin[9] for image processing, in particular because it processes CMYK images that PDFBox does not.

Some areas of work are outside the current scope of PDFBox, including the implementation of rendering for complex scripts. There is some provision, and some developers have contributed code for non-European languages where they have expertise (for example Users-5). In some cases the layout of the languages is sufficiently close to Latin scripts that there is no need for additional provision, if the fonts are correct as shown in PDFBOX-3293. However, for many languages including Arabic and those from the Indian subcontinent there is a need to implement code to position the glyphs using GSUB and GPOS tables. In PDFBOX-4189 a user provides a lot of the functionality to support GSUB tables for Bengali. The complexity of the task is clear from the discussions reviewing and accepting the source code.

Decisions are also made about the cause of observations and whether what is observed is the result of a problem with PDFBox. Where the issue lies with PDFBox, decisions are then made about resolving the problem. Sometimes the erroneous observation results from other software. A user reports a difference between the assessments by Adobe preflight and PDFBox concerning a document's compliance with the PDF/A-1b standard in PDFBOX-4045. Adobe XI identifies inconsistencies in the glyph widths for one font in the document. After investigation the core developers determine that there is no error in PDFBox and that Adobe X agrees that the document is compliant. Given the inconsistent assessments made by Adobe X and XI, and that inspection of the font does not show the issue reported by Adobe XI, the PDFBox core developers conclude there is a problem with the implementation of preflight in the particular version of Adobe XI used.

[9] https://github.com/haraldk/TwelveMonkeys.

Table 6
Thematic factors influencing software development decisions in the Apache PDFBox project.

Factor	Description
Workforce	The availability of contributors to do work.
Maintenance Risk	The maintenance burden for the project of a feature implementation.
Expertise	The collective expertise of the contributors to the project.
Sustainable Solution	The long-term viability of a technical solution.
Capability	The ability to make relevant and meaningful changes in a given context.
Intellectual Property Rights	Matters pertaining to copyright, patents and licensing.
Java Interoperability	The consequences for interoperability of revisions to Java.

4.2. Factors influencing decision-making

Common to the decision types observed is a set of considerations or factors that influence the outcome of the decision-making process (see Table 6).

4.2.1. Workforce

Companies choose to use the PDFBox software and, where appropriate for their needs, contribute to its improvement through the work of their developers. As noted, the core developers of PDFBox are few in number and are, as they emphasise, not paid for their work on PDFBox:

> "The project is a volunteer effort and were always looking for interested people to help us improve PDFBox. There are a multitude of ways that you can help us depending on your skills." (Apache PDFBox, 2019)

With limited time available to them (Targett, 2019), the PDFBox core developers concentrate their efforts (Khudairi, 2019) in areas of the software where work is a priority, unless other developers in the community are able to contribute.

The example given previously of work on a solution for a document merging problem (PDFBOX-3875 [10]) that halts may be explained by the limited workforce being focused on other, more achievable tasks, as illustrated by a core developers' comment on another task:

> "I had hoped to implement that but given current commitments I have it is unlikely that I'm able to do it in the short term (I'm trying to concentrate on resolving AcroForms related stuff in my spare time for the momen[t])." (PDFBOX-3550)

Another example of the influence of the available workforce on decision making can be found in PDFBOX-3875 where a developer working for a company wants a problem resolved. The problem is challenging and will take time to understand and resolve. The developer reporting the problem is given three choices: to adopt and use another OSS application, and, implicitly, to buy a licence for Adobe professional, or to contribute the fix themselves either directly or by commissioning other developers to do the work.

4.2.2. Maintenance risk

The notion of a maintenance risk can be related to the factors of expertise and workforce. Core developers will sometimes express or imply a concern that they are unwilling to accept a solution. For example, PDFBOX-3962 where a user proposes a solution that repairs the unicode mappings in one PDF document so that it can be

[10] Issue tracker tickets referenced in Section 4.2 are given in Table 7.

Table 7
Apache PDFBox Jira issue tracker tickets referenced in Section 4.2.

Factor	Issue tracker ticket
Workforce	PDFBOX-3550
	PDFBOX-3875
Maintenance risk	PDFBOX-3550
	PDFBOX-3962
Expertise	PDFBOX-3550
	PDFBOX-3844
	PDFBOX-4024
	PDFBOX-4095
	PDFBOX-4189
	PDFBOX-4267
Sustainable solution	PDFBOX-3300
Capability	PDFBOX-3641
Intellectual property rights	PDFBOX-3618
	PDFBOX-4320
Java interoperability	PDFBOX-3549

rendered. The core developers identify that the solution resolves a special case, and that further work would be required to develop a viable solution for the Java 9 libraries. Another concern articulated in some requests for support for complex scripts is that the core developers do not have the skills to maintain the functionality. A lengthy discussion of the issue can be found in PDFBOX-3550 where the core developers identify some central challenges to creating a solution. The main concern in both cases is that by providing additional functionality that cannot be maintained or is a challenge to maintain, either in terms of the effort required or the necessary expertise, there is a risk to the utility of the software, and, perhaps, the viability of the project.

4.2.3. Expertise

The implementation of PDF software requires expertise in a wide range of areas in addition to PDF itself. Limitations to the available expertise help determine what work can be done by contributors. One implication, already noted, is the reluctance to maintain source code in areas where there is no or limited expertise amongst the core developers. Another is that some areas of functionality cannot be developed. For example, a user asks about compressing CMYK JPEG images in PDFBOX-3844. The core developer responds by saying:

> "There is no JPEG compression from CMYK BufferedImage objects out of the box, i.e. Java ImageIO doesn't support it, and we don't have the skills, so that I'll have to close as "won't fix" this time." (PDFBOX-3844)

The alternative suggested in PDFBOX-3844 is to investigate the Twelve Monkeys project that builds on the Java ImageIO functionality.

There is also a great deal of expertise within the PDFBox community which can enable the implementation of solutions. In PDFBOX-4095 one contributor provides a proposed solution to a challenging problem. After some work evaluating the proposed change, which isn't going well, another contributor suggests a simple revision that resolves the problems. Similarly a complex image rendering problem is solved with the help of advice from a contributor in PDFBOX-4267, and another contributor implements code to process YCbCr CMYK JPEG images in PDFBOX-4024.

Expertise alone, however, is not sufficient to provide a solution to a problem in all cases. The discussion in PDFBOX-4189 shows there is considerable expertise within the user community and the core developers about fonts and how to render complex scripts. Key factors that have prevented the work being done previously have been not only a shortage of available workforce, but also a lack of expertise in the target language that would provide sufficient understanding to distinguish between good and bad solutions:

> "Many complex scripts (such as Arabic) require shaping engines which require deep knowledge of the languages in order to follow the rules in the OpenType tables." (PDFBOX-3550)

4.2.4. Sustainable solution

There are often implementation choices to be made when resolving a problem. The better long-term solution is more viable than the short-term fix, or workaround. In PDFBOX-3300 concerns are reported about the way that a font subset has been created prior to embedding it in a document. A specific solution is proposed that provides a way of resolving the problem. Another developer identifies that the optimal solution is to resolve some problems in the CMap[11] parser. It is a more sustainable solution than a patch to provide a specific workaround. In this case the developers are able to create a generic solution that better addresses the font standards, and thereby the PDF standards, and provides a longer-lived solution.

4.2.5. Capability

A key factor in decisions concerns whether the project is able to correct the problem that is causing the observed behaviour. The examples given in Section 4.1.2 where the PDFBox implementation was degraded from meeting the standard to match the behaviour of Adobe's software illustrate one aspect of capability as a factor. In those cases the 'incorrect' implementation could not be revised, and only a revision to PDFBox could ensure documents created would be rendered as expected by Adobe's implementations. In other cases bugs are found in external libraries or infrastructure that have an impact on PDFBox. Often a workaround will be found, or an alternative library recommended. For example, PDFBOX-3641 describes a situation in which PDFBox uses a core Java library in a way that triggers a bug in the Java implementation. The code in PDFBox is revised to prevent the bug being triggered. The Java bug is also reported[12].

4.2.6. Intellectual property rights

PDF documents can include technologies and artifacts where use is constrained by copyright, patents or licences. In addition, PDFBox is implemented in Java which during its lifetime has moved from closed source, to largely open source, to some variants (e.g. OpenJDK and derivatives like Amazon Corretto) that are entirely open source. An implementation of the JPEG 2000 codec was included in extensions to the Java libraries. During Sun Microsystems' process to make Java open source the codec along with other image codecs was released as a separate library known as ImageIO. The licence used for the implementation of the JPEG 2000 codec is not an Open Software Initiative (OSI) approved open source licence and some consider the licence used is incompatible with OSS licences such as the GPL v3 and the Apache Licence v2.0.[13] In addition there are concerns amongst OSS developers about the potential of patent claims related to JPEG 2000, though the concerns are diminishing with the passage of time. Most of the image codecs in the ImageIO library have been reincorporated into the Java libraries in OpenJDK since v9, but the JPEG 2000 codec has not. Consequently, JPEG 2000 support in PDFBox, where it is required by users, relies on the jai-imageio[14] implementation of the codec,

[11] A CMap is a table in a font file that maps character encodings to the glyphs that represent them.

[12] https://bugs.openjdk.java.net/browse/JDK-8175984.

[13] For example the opinion expressed at: https://github.com/jai-imageio/jai-imageio-jpeg2000.

[14] https://github.com/jai-imageio/jai-imageio-jpeg2000.

which is no longer maintained. A user reports using the OpenJPEG[15] implementation of JPEG 2000 in PDFBOX-4320. However, OpenJPEG is implemented in C and can only be used as native code and which may not be suitable for some deployment contexts. The development of a replacement OSS JPEG 2000 codec is inhibited by the resources, including expertise and finance, required to implement a large and complex standard.[16]

The ISO 19005-1:2005 standard (ISO, 2005, p. 11) for archival PDF documents mandates the embedding of fonts, including the standard 14 fonts,[17], or substitute fonts, in files so that the document contains all the resources required to render it. The requirement is stated as: "Only fonts that are legally embeddable in a file for unlimited, universal rendering shall be used." (ISO, 2005, p. 10). The requirement can be problematic because many fonts have licences that do not permit redistribution. The matter is discussed in PDFBOX-3618. The legality of the embedded fonts is the responsibility of the document creator. Both the PDF/A-1 and PDF/A-2 standards include a note that clarifies the need for the legal use of any font to be clearly and verifiably stated:

> "This part of ISO 19005 precludes the embedding of font programs whose legality depends upon special agreement with the copyright holder. Such an allowance places unacceptable burdens on an archive to verify the existence, validity and longevity of such claims." (ISO, 2005, p. 11; ISO, 2011, p. 15).

4.2.7. Java interoperability

In addition there are a set of problems concerning interoperability with Java that are an influence on the solutions implemented in PDFBox. Some are related to the PDF standards where Java is used to provide support such as image processing required by the standards. An example is found in PDFBOX-3549 where Java versions have differing capability to process ICC colour spaces, and in some versions there are bugs that affect the handling of ICC colour spaces. During the period of PDFBox activity investigated three new major versions of Java were released, and many revisions made to each version. There is also some evidence in the mailing lists and Jira tickets that some users are still using Java 5, which was already obsolete at the start of the period investigated.

4.3. Summary

Through analysis of two years of activity in the PDFBox project related to implementation of the PDF specifications and standards, we have identified four decision types related to development of the project software and seven factors that influence those decisions. The four decision types are related to adapting the software to emulate the behaviour of Adobe's PDF software, implementing the PDF standards, and the scope of the PDFBox implementation. The seven factors act in combination to facilitate and constrain development activity, especially the interplay between expertise and workforce.

5. Analysis

Much of the work of PDFBox contributors consists of trying to match the implementation of Adobe PDF reader software. The reasons for matching Adobe implementations are mostly clear, yet trying to emulate Adobe's software is clearly challenging, and solutions, including validators, that might reduce the extent of the challenges, and the risks, are themselves challenging to create.

5.1. The challenges of developing PDF parsers

The PDF specifications and standards specify that PDF software *may* try to reconstruct files where the cross reference table is incorrect or has been omitted. In practice the Principle of Robustness is applied in Adobe's PDF software so that PDF files that are not well-formed can often be rendered. The developers of other PDF applications are obliged to follow Adobe's lead. If the developers of non-Adobe PDF software did not implement parsers that behaved similarly to Adobe's then their products would quickly become irrelevant because PDF users often believe that because documents can be read and rendered by Adobe software that they must meet the standard (Amiouny, 2016; Lehtonen et al., 2018). The extent to which PDF applications and libraries are expected to tolerate errors in documents is documented by Adobe's software, which creates a number of challenges for developers of PDF software.

Firstly, non-Adobe developers are left with the time-consuming puzzle of trying to match the Adobe implementations. Indeed, the puzzle includes an element of chance because differences in performance are discovered when a PDF document including a triggering problem is processed. Secondly, there are clearly security concerns in this approach. Parsing is arguably one of the more challenging software engineering tasks. In the case of PDF, the core specifications and standards are extensive and complex, and include a large number of normative references for component file and media types, all of which need to be parsed by either a PDF implementation or its dependencies. PDFBox has been the subject of Common Vulnerabilities and Exposures (CVE) notices related to parser implementation[18], as have other PDF software implementations. The core developers are therefore making decisions about security as part of those around the viability of the software when trying to match the behaviour of Adobe's software.

Some practitioners argue that a small revision made in the ISO 32000-2:2017 standard concerning the structure of the file that more precisely defines the relationship between the header and the end of file marker largely put an end to the need to apply the Principle of Robustness in PDF parsing (Amiouny, 2017). However, though the changes in the standard are important and may ease some of the burden on developers, we do not share the optimism because the changes only apply to structure of documents that are or claim to be PDF v2.0 compliant. Of course, there remain in circulation all the documents created during some 25 years of PDF usage, as well as those documents that will continue to be created which are compliant with earlier specifications and standards. Further, the Principle of Robustness is applied to tolerate nonconformance with normative standards of PDF, such as fonts and images, as well as minor PDF implementation errors. Given the history of malformed PDF files and the challenges of standards compliance, the fact that a document claims to be PDF v2.0 and complies with the structural requirements of ISO 32000-2:2017 does not guarantee that either the document or its components comply with the standard. Consequently, the need for tolerant parsing remains.

One improvement might be the creation of reference implementations and validation tools; practices that have been adopted in the development of open standards, for example in the IoT domain as noted in Section 2.3 (e.g. Watteyne et al., 2016). Validation tools for fonts could help ensure that font creators build font files that contain sufficient, accurate information for other software to use the font file, and that implementers of font parsers have a means by which to evaluate their software. Further, validation tools for PDF documents and a reference implementation for PDF would help the developers of PDF software create more interoperable ap-

[15] http://www.openjpeg.org/.

[16] JPEG 2000 is defined in ISO/IEC 15444 which consists of 14 parts (see Lundell et al., 2018).

[17] PDF specifications require 14 fonts to be present on systems that render documents, e.g. ISO 32000-1:2008 (ISO, 2008, p. 256).

[18] For example CVE-2018-8036 and CVE-2018-117979.

plications, with less effort and, possibly, reduce the security risks arising from the need to parse malformed documents. However, in practice PDF validators are difficult and expensive to implement. The veraPDF (veraPDF, 2019) PDF/A validator, for example, was created during a European Union funded project, and the PDFTools validator is proprietary licenced software.[19] The problem remains, also, that solutions such as validators are forward looking, and can not address the challenge of processing non-compliant PDF files created during the last 25 years that still need to be read. There is, though, a case for introducing validators and reference implementations to help ensure that PDF files created in the future pose fewer problems for software developers (Lundell and Gamalielsson, 2018). Furthermore, tools such as validators provide a reference point against which to try to improve the quality of existing documents, exemplified by the work of Lehtonen et al. (2018) with applications in PDF file preservation.

5.2. Practice vs standard

Other challenges for PDFBox contributors arise from the development of practice, particularly by Adobe, and where that moves away from the standards. PDFBOX-3913 records the discovery that Adobe's PDF software and PDF.js exceed the ISO 32000-1:2008 standard by implementing UTF-16 encoding for destination URIs in links. The bug report dates from August 2017 and is contemporary with the publication of ISO 32000-2:2017, which specifies the use of UTF-8 encoding (ISO, 2017, p. 515). Given the use of UTF-16 encoded URIs, which have been part of HTML 5 since 2011,[20], it is outwardly reasonable for Adobe and others to follow practice. However, it remains an open question why UTF-16 encoding for URIs was not part of the ISO 32000-2:2017 standard.

A further issue found in some PDFBox Jira issue tickets is a grey area between the standard and how a document is presented. The PDF specifications and standards apply to the quality of the document, and the manner in which some parts of the document are to be rendered (for example character spacing). However, the standard does not specify how software might render all of the document. The examples given above to illustrate degradation of compliance with the standard to match Adobe's implementation are of particular interest. The ISO 32000-1:2008 and ISO 32000-2:2017 standards are clear on how the value of the border width should be represented in a compliant PDF document. As the PDFBox core developers identified, the representation of the values of border widths within the document does not comply with the PDF specifications and standards because valid non-integer values are not accepted by Adobe software. However, the presentation on screen by Adobe software of border widths defined in the document is an interpretation of the values in the document, and one that may not need to be followed slavishly.

5.3. Project sustainability

The PDFBox core developers generally act to improve the functionality of the project software. However, there are times when their actions appear to be constrained by the long-term interests of the project. Some decisions, for example around the support for complex scripts and graphics processing, have ready explanations in that the core developers do not always have the necessary skills, or time, to implement the required solutions. There are also some activities where there may not be a clear decision stated, but the core developers, and some other contributors, do not complete tasks because they have run out of time, or have other, higher priority, tasks to attend to. It may be inferred that the developers are acting in the long-term interests of the project to create software that works and can be maintained. The concern being that if the project contributors overreach their collective abilities and their capacity to develop and maintain good quality software, then there is a risk the project *may* cease to be viable. There are parallels to be drawn between the decision-making of the core developers where they reflect their capacity to make and maintain specific changes and the decisions made within a business to maintain itself as a going concern. Implicit is the idea that the PDFBox software remains marketable, i.e. that the software is sufficiently compliant with the PDF specifications and standards that it is useful to many users, and the project will therefore continue to attract users and contributors without the need to take risks by making unsustainable changes.

It should be recognised that this observed process of self-regulation is precisely that. There is no company or group of companies driving the development of PDFBox and making strategic decisions. There are no dedicated managers making strategic decisions. Instead, what appear to be sensible, level-headed strategic decisions that might be made by a business are being made in the small by a small collective of individuals and company developers collaborating on the development and maintenance of PDFBox.

5.4. Limitations

The case study reported in this article describes and analyses the activity of practitioners collaborating in an OSS community to develop software that can create and process PDF documents. We acknowledge the limitations to the transferability of our findings that arise from the nature of the study. However, we conjecture that the findings may be representative of the challenges faced and decision types made in other OSS projects and, perhaps businesses, implementing standards-based interoperable software, in particular where a dominant implementation contributes to the discourse on the meaning of interoperability. Further, the factors informing the decisions made relate to technical and resource concerns that appear to be relevant for other businesses and organisations.

6. Conclusions

The study reports findings from an investigation of the practical decisions concerning interoperability made during a two year period by contributors to a community open source software project (Apache PDFBox). The PDFBox project develops and maintains software that can be used to create and process documents that conform to multiple PDF specifications, some of which have been published as ISO standards. Four types of decision made by contributors to maintain the interoperability of the PDFBox software were identified through thematic analysis. Decisions on software interoperability concern compliance with the PDF specifications and ISO standards, and to match or mimic the behaviour of the *de facto* reference implementation, where that is unrelated to the standards or in conflict with them. In conjunction, contributors also make decisions about the scope of the PDFBox implementation. Contributors to the PDFBox project are able to deliver high quality software through a careful, and at times, conservative, decision-making process that allows an often agile response to the discovery of problems with the project's software and to changes in the dominant proprietary implementation. At the same time, the decisions made are informed by factors including resource and technical considerations which contribute towards the longer term viability of the project and the software created.

[19] PDFTools 3-Heights Validator: https://www.pdf-tools.com/pdf20/en/products/pdf-converter-validation/pdf-validator/.

[20] https://www.w3.org/TR/2011/WD-html5-20110525/urls.html.

In summary, the study makes the following contributions to the existing body of knowledge in this area:

- A rich and detailed account of types of decisions made within a community OSS project to maintain software interoperability;
- An account of technical and non-technical factors that motivate and constrain software development activity in the project and support project sustainability.

This study provides a rich illustration and analysis of the challenges faced by contributors to a community OSS project to implement and maintain interoperable, standards-based software. The study has shown how the contributors to PDFBox are able to meet challenges arising from the demands of the technical specifications and standards, and the performance of a *de facto* reference implementation. The study also finds that through awareness of the resources available to the project, the project is able to maintain interoperable software of continuing technical relevance. A topic for future research is to understand the extent to which the challenges and the decision-types identified, and the factors influencing those decisions are representative of those faced by other organisations — businesses and OSS projects — developing standards-based implementations.

Declaration of competing interest

None.

Acknowledgements

This research has been financially supported by the Swedish Knowledge Foundation (KK-stiftelsen) and participating partner organisations in the LIM-IT project. The authors are grateful for the stimulating collaboration and support from colleagues and partner organisations.

References

Ahlgren, B., Hidell, M., Ngai, E.C.H., 2016. Internet of things for smart cities: interoperability and open data. IEEE Internet Comput. 20, 52–56. doi:10.1109/MIC.2016.124.

Allman, E., 2011. The robustness principle reconsidered. Commun. ACM 54, 40–45. doi:10.1145/1978542.1978557.

Amiouny, D., 2016. Buggy PDF Files, Should We Try to Fix Them?. Amyuni Technologies Inc.. http://blog.amyuni.com/?p=1627. Accessed: 2019-05-15.

Amiouny, D., 2017. PDF 2.0 and the Future of PDF: Takeways from PDF Days Europe 2017. Amyuni Technologies Inc.. http://blog.amyuni.com/?p=1702. Accessed: 2019-05-14.

Apache PDFBox, 2019. Apache PDFBox: a Java PDF Library. The Apache Software Foundation. https://pdfbox.apache.org/. Accessed: 2019-09-17.

Apache Tika, 2019. Apache Tika — a Content Analysis Toolkit. Apache Software Foundation. https://tika.apache.org/. Accessed: 2019-06-05.

ASF, 2019. The Apache Software Foundation. The Apache Software Foundation. http://apache.org/. Accessed: 2019-06-05.

ASF, 2019. Apache Software Foundation Public Mailing List Archives. Apache Software Foundation. http://mail-archives.apache.org/. Accessed: 2019-06-05.

Atlassian, 2019. Jira REST APIs. Atlassian. https://developer.atlassian.com/jiradev/jira-apis/jira-rest-apis. Accessed: 2019-04-15.

Bitergia, 2019. GrimoireLab. Bitergia. https://chaoss.github.io/grimoirelab/. Accessed: 2019-06-03.

Black Duck, 2019. Apache PDFBox. Black Duck Software Inc.. https://www.openhub.net/p/pdfbox/. Accessed: 2019-03-08.

Bogk, A., Schöpl, M., 2014. The pitfalls of protocol design: attempting to write a formally verified PDF parser. In: 2014 IEEE Security and Privacy Workshops, pp. 198–203. doi:10.1109/SPW.2014.36.

Bouvier, D.J., 1995. Versions and standards of HTML. SIGAPP Appl. Comput. Rev. 3, 9–15. doi:10.1145/228228.228232.

Bradner, S., 1996. The internet standards process — revision 3. Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc2026.html. Accessed: 2019-09-19.

Bradner, S., 1999. The internet engineering task force. In: DiBona, C., Ockman, S., Stone, M. (Eds.), Opensources: Voices from the Open Source Revolution. O'Reilly & Associates, pp. 28–30.

Braun, V., Clarke, V., 2006. Using thematic analysis in psychology. Qual. Res. Psychol. 3, 77–101. doi:10.1191/1478088706qp063oa.

Butler, S., Gamalielsson, J., Lundell, B., Brax, C., Sjöberg, J., Mattsson, A., Gustavsson, T., Feist, J., Lönroth, E., 2019. On company contributions to community OSS projects. IEEE Trans. Softw. Eng. (early access) doi:10.1109/TSE.2019.2919305. 1–1.

CEF Digital, 2019. Start Using Digital Signature Services (DSS). CEF Digital. https://ec.europa.eu/cefdigital/wiki/pages/viewpage.action?pageId=71776034. Accessed: 2019-04-29.

Davies, E.B., Hoffmann, J., 2004. IETF Problem Resolution Process. Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc3844.html. Accessed: 2019-09-19.

De Coninck, Q., Michel, F., Piraux, M., Rochet, F., Given-Wilson, T., Legay, A., Pereira, O., Bonaventure, O., 2019. Pluginizing QUIC. In: Proceedings of the ACM Special Interest Group on Data Communication. ACM, New York, NY, USA, pp. 59–74. doi:10.1145/3341302.3342078.

Eclipse Foundation, 2019. Californium (Cf) CoAP framework. Eclipse Foundation. https://www.eclipse.org/californium/. Accessed: 2019-10-03.

Eclipse Foundation, 2019. Eclipse Leshan. The Eclipse Foundation. https://www.eclipse.org/leshan/. Accessed: 2019-10-03.

Eclipse Foundation, 2019. Eclipse Wakaama. The Eclipse Foundation. https://www.eclipse.org/wakaama/. Accessed: 2019-10-03.

Eclipse IoT Working Group, 2019. Open Source for IoT. Eclipse IoT Working Group. https://iot.eclipse.org/. Accessed: 2018-08-29.

Egyedi, T.M., 2007. Standard-compliant, but incompatible?!. Comput. Standards Interfaces 29, 605–613. doi:10.1016/j.csi.2007.04.001.

Endignoux, G., Levillain, O., Migeon, J.Y., 2016. Caradoc: A pragmatic approach to PDF parsing and validation. In: 2016 IEEE Security and Privacy Workshops (SPW), pp. 126–139. doi:10.1109/SPW.2016.39.

Fitzgerald, B., 2006. The transformation of open source software. Manage. Inf. Syst. Q. 30, 587–598.

Gamalielsson, J., Lundell, B., 2013. Experiences from implementing PDF in open source: Challenges and opportunities for standardisation processes. In: Proceedings of the 8th International Conference on Standardization and Innovation in Information Technology (SIIT) 2013, pp. 1–11. doi:10.1109/SIIT.2013.6774572.

Gerring, J., 2017. Case Study Research: Principles and Practices, second ed. Cambrige University Press, Cambridge, UK.

ICIJ, 2019. The Panama Papers: Exposing the Rogue Offshore Finance Industry. https://www.icij.org/investigations/panama-papers/. Accessed: 2019-05-29.

IETF, 2019. Internet Engineering Task Force. Internet Engineering Task Force. https://www.ietf.org/. Accessed: 2019-09-27.

IETF, 2019. QUIC (quic) – about. Internet Engineering Task Force. https://datatracker.ietf.org/wg/quic/about/. Accessed: 2019-09-24.

IETF, 2019. QUIC (quic) – documents. Internet Engineering Task Force. https://datatracker.ietf.org/wg/quic/documents/. Accessed: 2019-09-24.

ISO, 2005. Document management – Electronic Document File Format for Long-Term Preservation – Part 1: Use of PDF 1.4 (PDF/A-1) (ISO 19005-1:2005), first ed. International Organization for Standardisation, Geneva, Switzerland.

ISO, 2008. Document Management – Portable Document Format – Part 1: PDF 1.7 (ISO 32000-1:2008), first ed. International Organization for Standardisation, Geneva, Switzerland.

ISO, 2011. Document Management – Electronic Document File Format for Long-Term Preservation – Part 2: Use of ISO 32000-1 (PDF/A-2) (ISO 19005-2:2011), first ed. International Organization for Standardisation, Geneva, Switzerland.

ISO, 2013. Digital Compression and Coding of Continuous-Tone Still Images: JPEG File Interchange Format (JFIF) (ISO/IEC 10918-5:2013), first ed. International Organization for Standardisation, Geneva, Switzerland.

ISO, 2017. Document Management – Portable document format – Part 2: PDF 2.0 (ISO 32000-2:2017), first ed. International Organization for Standardisation, Geneva, Switzerland.

JPEG, 2019. Overview of JPEG XT. International Standards Organisation. https://jpeg.org/jpegxt/. Accessed: 2019-04-01.

Kelly, M., Nelson, M.L., Weigle, M.C., 2014. The archival acid test: Evaluating archive performance on advanced HTML and JavaScript. In: IEEE/ACM Joint Conference on Digital Libraries, pp. 25–28. doi:10.1109/JCDL.2014.6970146.

Khudairi, S., 2017. The Apache Software Foundation Recognizes Apache Innovations Integral to the Pulitzer Prize-winning Panama Papers investigation. Apache Software Foundation. https://blogs.apache.org/foundation/entry/the-apache-software-foundation-recognizes. Accessed: 2019-02-14.

Khudairi, S., 2019. Apache in 2018 — by the Digits. Apache Software Foundation. https://blogs.apache.org/foundation/entry/apache-in-2018-by-the. Accessed: 2019-01-02.

Ko, J., Eriksson, J., Tsiftes, N., Dawson-Haggerty, S., Vasseur, J., Durvy, M., Terzis, A., Dunkels, A., Culler, D., 2011. Industry: Beyond Interoperability: Pushing the Performance of Sensor Network IP Stacks. In: Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems. ACM, New York, NY, USA, pp. 1–11. doi:10.1145/2070942.2070944.

Lehmkühler, A., 2010. Apache PDFBox — Working with PDFs for Dummies. The Apache Software Foundation. https://people.apache.org/lehmi/apachecon/ApacheConPDFBox.pdf. Accessed: 2019-06-04.

Lehtonen, J., Helin, H., Kylander, J., Koivunen, K., 2018. PDF mayhem: is broken really broken? In: Proceedings of the 15th International Conference on Digital Preservation (iPRES 2018) doi:10.17605/OSF.IO/FZXC9.

Lindlar, M., Tunnat, Y., Wilson, C., 2017. A test-set for well-formedness validation in JHOVE — the good, the bad and the ugly. In: Proceedings of the 15th International Conference on Digital Preservation (iPRES 2017) doi:10.5281/zenodo.1228649.

Lundell, B., 2011. e-Governance in public sector ICT procurement: what is shaping practice in Sweden? Eur. J. ePractice 12, 66–78. https://joinup.ec.europa.eu/sites/default/files/document/2014-06/ePractice%20Journal-%20Vol.%2012-March_April%202011.pdf.

Lundell, B., Gamalielsson, J., 2017. On the potential for improved standardisation through use of open source work practices in different standardisation organisations: how can open source-projects contribute to development of IT-standards? In: Blind, K., Jakobs, K. (Eds.) Digitalisation: Challenge and Opportunity for Standardisation. Proceedings of the 22nd EURAS Annual Standardisation Conference, EURAS Contributions to Standardisation Research, Vol. 12. Verlag Mainz, Aachen, pp. 137–155.

Lundell, B., Gamalielsson, J., 2018. Sustainable digitalisation through different dimensions of openness: How can lock-in, interoperability, and long-term maintenance of IT systems be addressed? In: Proceedings of OpenSym '18. ACM, New York, NY, USA doi:10.1145/3233391.3233527.

Lundell, B., Gamalielsson, J., Katz, A., 2018. On challenges for implementing ISO standards in software: Can both open and closed standards be implemented in open source software? In: Jakobs, K. (Ed.) Corporate and Global Standardization Initiatives in Contemporary Society. IGI Global, Hershey, PA, USA, pp. 219–251. doi:10.4018/978-1-5225-5320-5.

Lundell, B., Gamalielsson, J., Tengblad, S., Yousefi, B.H., Fischer, T., Johansson, G., Rodung, B., Mattsson, A., Oppmark, J., Gustavsson, T., Feist, J., Landemoo, S., Lönroth, E., 2017. Addressing lock-in, interoperability, and long-term maintenance challenges through open source: How can companies strategically use open source? In: Open Source Systems: Towards Robust Practices - Proceedings of the 13th IFIP WG 2.13 International Conference on Open Source Systems, OSS 2017. Springer, pp. 80–88. doi:10.1007/978-3-319-57735-7_9.

Mladenov, V., Mainka, C., Meyer zu Selhausen, K., Grothe, M., Schwenk, J., 2018a. 1 Trillion dollar refund — how to spoof PDF signatures. https://www.pdf-insecurity.org/download/paper.pdf. Accessed: 2019-05-09.

Mladenov, V., Mainka, C., Meyer zu Selhausen, K., Grothe, M., Schwenk, J., 2018b. How to break PDF signatures. https://pdf-insecurity.org/. Accessed: 2019-05-14.

Nikolich, P., I, C. L., Korhonen, J., Marks, R., Tye, B., Li, G., Ni, J., Zhang, S., 2017. Standards for 5G and beyond: their use cases and applications. https://futurenetworks.ieee.org/tech-focus/june-2017/standards-for-5g-and-beyond. Accessed: 2019-10-03.

OMA, 2019. OMA SpecWorks. Open Mobile Alliance. https://www.omaspecworks.org/. Accessed: 2019-10-03.

Patton, M.Q., 2015. Qualitative Research and Evaluation Methods, fourth ed. Sage Publications Inc., Thousand Oaks, California, USA.

Phillips, B., 1998. Designers: the browser war casualties. Computer 31, 14–16. doi:10.1109/2.722269.

Phipps, S., 2019. Open Source and FRAND: Why Legal Issues are the Wrong Lens. Open Forum Academy. http://www.openforumeurope.org/wp-content/uploads/2019/03/OFA_-_Opinion_Paper_-_Simon_Phipps_-_OSS_and_FRAND.pdf. Accessed: 2019-10-03.

Piraux, M., De Coninck, Q., Bonaventure, O., 2018. Observing the evolution of QUIC implementations. In: Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC. ACM, New York, NY, USA, pp. 8–14. doi:10.1145/3284850.3284852.

Postel, J., 1981. RFC 793: Transmission Control Protocol. Internet Engineering Task Force. https://tools.ietf.org/html/rfc793. Accessed: 2019-04-15.

Richter, T., Clark, R., 2018. Why JPEG is not JPEG — testing a 25 years old standard. In: 2018 Picture Coding Symposium (PCS), pp. 1–5. doi:10.1109/PCS.2018.8456260.

Riehle, D., 2011. Controlling and steering open source projects. IEEE Comput. 44, 93–96. doi:10.1109/MC.2011.206.

Rossi, B., Russo, B., Succi, G., 2008. Analysis about the diffusion of data standards inside European public organizations. In: 2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications, pp. 1–6. doi:10.1109/ICTTA.2008.4529953.

Shelby, Z., Hartke, K., Bormann, C., 2014. The Constrained Application Protocol (CoAP). Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc7252.html. Accessed: 2019-10-03.

Targett, E., 2019. Meet the Apache Software Foundations Top 5 code Committers. Computer Business Review. https://www.cbronline.com/feature/apache-top-5. Accessed: 2019-10-04.

The Document Foundation, 2019. LibreOffice. The Document Foundation. https://www.libreoffice.org/. Accessed: 2019-09-26.

Treese, W., 1999. Putting it together: Engineering the Net: The IETF. netWorker 3, 13–19. doi:10.1145/294626.294634.

veraPDF, 2019. Industry supported PDF/A validation. veraPDF Consortium. http://verapdf.org/. Accessed: 2019-06-03.

W3C, 2019. The history of the web. World Wide Web Consortium. https://www.w3.org/wiki/The_history_of_the_Web. Accessed: 2019-09-18.

W3C, 2019. World wide web consortium (W3C). World Wide Web Consortium. https://www.w3.org/. Accessed: 2019-09-18.

Walsham, G., 2006. Doing interpretive research. Eur. J. Inf. Syst. 15, 320–330. doi:10.1057/palgrave.ejis.3000589.

WaSP, 2019. History of the Web Standards Project. The Web Standards Project. https://www.webstandards.org/about/history/. Accessed: 2019-09-27.

Watteyne, T., Handziski, V., Vilajosana, X., Duquennoy, S., Hahm, O., Baccelli, E., Wolisz, A., 2016. Industrial wireless IP-based cyber-physical systems. Proc. IEEE 104, 1025–1038. doi:10.1109/JPROC.2015.2509186.

Wilson, C., McGuinness, R., Jung, J., 2017. veraPDF: Building an open source, industry supported PDF/A validator for cultural heritage institutions. Digital Lib. Perspect. 33, 156–165.

Wilson, J., 1998. The IETF: Laying the Net's asphalt. Computer 31, 116–117. doi:10.1109/2.707624.

Wright, S.A., Druta, D., 2014. Open source and standards: the role of open source in the dialogue between research and standardization. In: 2014 IEEE Globecom Workshops (GC Wkshps), pp. 650–655. doi:10.1109/GLOCOMW.2014.7063506.

Simon Butler received a Ph.D. from The Open University in 2016. He is a researcher in the Software Systems Research Group at the University of Skövde in Sweden. His research interests include software engineering, open source software, program comprehension, software development tools and practices, and software maintenance.

Jonas Gamalielsson received a Ph.D. from Heriot Watt University in 2009. He is a senior lecturer at the University of Skövde and is a member of the Software Systems Research Group. He has conducted research related to free and open source software in a number of projects, and his research is reported in publications in a variety of international journals and conferences.

Professor Björn Lundell received a Ph.D. from the University of Exeter in 2001, and leads the Software Systems Research Group at the University of Skövde. Professor Lundell's research contributes to theory and practice in the software systems domain, in the area of open source and open standards related to the development, use, and procurement of software systems. His research addresses socio-technical challenges concerning software systems, and focuses on lock-in, interoperability, and longevity of systems. Professor Lundell is active in international and national research projects, and has contributed to guidelines and policies at national and EU levels.

Christoffer Brax received the M.Sc. degree from the University of Skövde in 2000, and a Ph.D. from Örebro University in 2011. He is a consultant with Combitech AB working in systems engineering, requirements management, systems design and architecture, and IT security. Christoffer has 18 years experience as a systems engineer.

Anders Mattsson received the M.Sc. degree from Chalmers University of Technology, Sweden, in 1989 and a Ph.D. in software engineering from the University of Limerick, Ireland in 2012. He has almost 30 years experience in software engineering and is currently R&D manager for Information Products and owner of the software development process at Husqvarna AB. Anders is particularly interested in strengthening software engineering practices in organizations. Special interests include software architecture and model-driven development in the context of embedded real-time systems.

Tomas Gustavsson received the M.Sc. degree in Electrical and Computer Engineering from KTH Royal Institute of Technology in Stockholm in 1994. He is co-founder and current CTO of PrimeKey Solutions AB. Tomas has been researching and implementing public key infrastructure (PKI) systems for more than 24 years, and is founder and developer of the open source enterprise PKI project EJBCA, contributor to numerous open source projects, and a member of the board of Open Source Sweden. His goal is to enhance Internet and corporate security by introducing cost effective, efficient PKI.

Jonas Feist received the M.Sc. degree in Computer Science from the Institute of Technology at Linköping University in 1988. He is senior executive and co-founder of RedBridge AB, a computer consultancy business in Stockholm.

Erik Lönroth holds an M.Sc. in Computer Science and is the Technical Responsible for the high performance computing area at Scania IT AB. He has been leading the technical development of four generations of super computing initiatives at Scania and their supporting subsystems. Erik frequently lectures on development of super computer environments for industry, open source software governance and HPC related topics.