Changing the way the world works with documents.

**6 Important Factors when Choosing a PDF Library**

**Adam Pez A PDF**

**DOCUMENT**

**Overview**

You plan to embed PDF functionality into an application. But before you dive into the project, you must decide: do you go with a more expensive commercial PDF SDK—or a lower-cost alternative such as an open-source library or an open-source wrapper?

There are non-trivial costs to switching later. Developers have to re-learn the new library, re-adjust the backend, customize the UI to match what users are

accustomed to, as well as migrate documents, form data, annotations, and more.

According to **market research** conducted by Stax Inc., the average Net Promoter Score (NPS) for the topﬁvePDF SDK vendors is 35%. And 70% of customers express interest in switching despite the high costs.

This dissatisfaction implies that picking the right PDF SDK is a lot harder than it seems. And to help you avoid the same mistakes as past implementations, we’ve written this article.

(We also **recently surveyed 57**unique organizations that switched from PDF.js to a commercial SDK. Read our **comprehensive guide to PDF.js** to learn more.)

**2**

**Restrictions on Features and Platforms**

A ﬁrst mistake organizations make when selecting a PDF library for theﬁrst time is to assume ﬁxed feature requirements. But these are likely to evolve.

Users start to ask for more functionality as they grow dependent upon a PDF SDK. An organization

will then have to consider saying no to user feature requests; building time-intensive and challenging customizations on top; or integrating additional libraries and thus adding more complexity, maintenance

overhead, and risk. Additionally, a library may workﬁneinitially on the main platforms preferred by your users. But later if you wish to expand, the library does not support the platforms you want—or the APIs are inconsistent, with diﬀerent classes and methods across platforms

making it so your engineers have to start from scratch.

To avoid this hidden cost, go with an SDK with a broad feature-set across

multiple platforms, providing you theﬂexibility to grow down the road.

Maybe big companies can absorb the costs of maintaining

three-to-four diﬀerent relationships with diﬀerent vendors, each with a diﬀerent code base, diﬀerent roadmaps, and diﬀerent problems.I’mnot saying it isn’t possible.

***Kalsefer Co-Founder and CEO, Avshi Segev***

**3**

**Unanticipated Diﬃculty Adding Features**

Another mistake is where

organizations select a basic library to save money with the assumption that they can build anything needed

on top.

But building in an unfamiliar domain can easily lead to unknown challenges, high expenses, and reduced speed to market. And PDF is unusually complex as high-proﬁle teams attest—including those of **Slack**, **Dropbox**, and **Linkedin**.

Your devs are not necessarily PDF experts, and attempting challenging PDF features in-house involves a steep learning curve not subject to economies of scale. Throwing more devs into the equation will not shrink the ramp-up time for the ﬁrst developer.

Additionally, custom features will have to be supported and maintained long-term, creating an additional

ongoing opportunity cost: committed resources will be less-available to work on other projects.

PDFs are complex documents—structured into

diﬀerent layers of information, data, and objects, and containing diﬀerent languages, images, and graphics.

***Developer, Slack***

PDF is an incredibly complexﬁleformat—the speciﬁcation is more than a thousand pages long, not including the extensions and supplements.

***Developer, Dropbox***

PDFs are an incredibly complexﬁleformat; this is especially so given that a PDF can be generated a hundred

diﬀerent ways, all of which a renderer needs to handle gracefully.

***Developer, LinkedIn***

**4**

Organizations that we’ve spoken to have found the most challenging features are

those that require engaging PDF at a low level, where objects are deﬁned in PDF byte code—with unique byte oﬀsets for

diﬀerent objects, making it diﬃcult for devs unfamiliar with PDF’s inner workings to parse and manage these objects correctly

Challenging PDF functionality includes

**•Managing PDF annotations from multiple users (e.g., synchronization and versioning**

**•PDF generation (creating PDFs from scratch or from other documents)**

**•Page manipulation (add, merge, or remove)**

**•Layers (via Optional Content Groups)**

**•Color management features (e.g., ink-color separations, overprint, etc.)**

While it is certainly possible to build the above in-house, PDF features can consume a shocking amount of time. And you eventually may have to decide whether to continue—or whether to bite the bullet and abandon months or years of work for an alternative that can meet your requirements cost-eﬀectively.

To avoid this type of hidden cost, you will want to carefully consider the capacity of your existing

development team should you decide to build,

maintain, and support custom PDF features inhouse as these features

often prove time-intensive and challenging.

...you shouldn’t build anything that’s availableoﬀthe shelf because it’s not a source of competitive advantage if everybody else can avail themselves of it. The only scenario where you should build is if it’s your core technology -- the core source of your competitive diﬀerentiation and competitive advantage.

***Mark Holst-Knudsen, President ThomasNet @MIT’s 2014 CIO***

**5**

A lower-quality library also encounters performance and memory issues, such as large documents with frustratingly long wait times for your users as well as complex documents that crash the viewer. This is often due to the absence of features such as PDF tiling, parallelization, and **linearization** that a more mature PDF SDK will incorporate.

Some solutions (e.g., image servers) perform excellently when tested on a small number of documents and

users but then inﬂict unexpected hidden costs when scaled up. When hundreds or thousands of users later view, mark up, comment on, and

otherwise interact with (i.e.,scroll, pan, and zoom) documents, server

resource and network data usage explodes. To maintain your desired UX, you have to pay higher fees or invest in more servers.

The following types of documents have much more demanding rendering requirements:

**• CAD-based PDFs such as construction**

**and engineering drawings with very large and complex designs.**

**• Reports, textbooks, and marketing material using advanced PDF graphics such as shadings, gradients, soft masks, and patterns.**

**• Geospatial maps with OCG layers that are switchedoﬀby default.**

**• Pre-press documents which require an SDK with advanced color management features to print colors accurately.**

**• High-speed accurate rendering**

**(especially on native mobile apps and mobile browsers).**

**• Context extraction of tables, text, etc. with document structure (e.g., text read**

**order or table arrangement) in tact.**

To prevent crashes, slowness, and rendering issues from disrupting your UX, test functionality with the types of documents your users will work with. Also test a

server-based solution at the anticipated load and usage.

**6**

**Poor UX: Slow Performance, Crashing, and Inaccurate Rendering**

Another source of hidden costs can be a poor user experience, especially as users start to upload more massive and complex documents that crash or freeze a lower-quality viewer. **Construction Computer** Software encountered these issues with a free PDF viewer add-on to its ﬂagship estimation software.

As is often the case with a lower-quality library, PDFs render incorrectly. You then have to wait on the vendor to respond. But a reseller or a smaller company with many remote developers may have diﬃculty providing the same turnaround time and specialized support and service as a commercial SDK. If they did not build the rendering engine themselves, they may not be able toﬁxthe issue—or ﬁxes may take a long time—because they have

diﬃculty ﬁnding in the code where the problem originated. If you go with

open-source, you may have toﬁxbugs yourself.

If you’re looking for a PDF reader for the ﬁrst time, you better make sure it can read 100% of your PDF ﬁles. Because if your client-base

starts relying on that PDF reader, exactly what happened to us, they still want the absolute best quality.”

***Tony Cornwall, Construction Computer Software***

**7**

**Low Adoption**

**on a Complex UI**

In 2018, AEC-software company PlanGrid partnered with FMI to survey nearly 600 construction leaders from around the world to discern why construction and engineering software succeed or fail. The ﬁndings **report** “Construction Disconnected” identiﬁed a complex UI and inadequate user training as two of the topﬁvereasons for why technology fails.

Being able to slim down the interface and tailor feature-sets to speciﬁc

user groups is proven to signiﬁcantly cut down training costs and improve user adoption. (See our **OEC Graphics success story** to learn

more.)

However, a closed-source UI will limit you in what you can customize, and you may not be able to fullyﬁxthe

UX. (And by the time you’ve discovered this, it may be too late.) A closedsource UI will make it diﬃcult to evaluate how deeply you can customize, optimize, and add new tools or annotation types to the UI. Therefore, your team may build out a proof of concept and make their plans for future expansion—only to have to scale back their ambitions or wait on the vendor to adjust the API. A black box UI will prove especially problematic if your UI team is very strict or if you have unique UI requirements (e.g., accessibility compliance requirements such as ADA/508).

To avoid this hidden cost, choose a vendor with an open-source UI or

make certain your proof of concept

**8**

**Security Issues**

When writing PDF features from scratch, developers may be tempted to take shortcuts to save time. But these shortcuts cause the solution to become obsolete quickly as devs run into the exact security issues a more mature toolkit makes a lot easier to solve. One recent instance our solution engineers have noted is where developers use JavaScript-based submit buttons on forms rather than uploading and parsing data out of forms—which opens up the system to phishing and middle-man attacks. Someone could easily edit the button to have it send personal information to another server, and then maliciously re-circulate the form within your organization or send it to end users.

**Vendor Lock-in**

Lastly, consider how your data and documents will be stored. For example, annotations stored in a proprietary format, such as Brava! annotations and some versions of JSON, will not be accessible to users who want to view their annotations with other tools such as Adobe Acrobat. Moreover, it will be challenging to migrate these annotations later if you wish to switch solutions. A vendor who manages annotations in the ISO standard for annotations interchange, XFDF, for example, will eliminate this hidden cost.

**The Bottom Line**

The best way to avoid hidden costs associated with the wrong PDF library is to perform due diligence during your evaluation. To assist you in this process, we’ve written a blog with several considerations you can add to your PDF SDK evaluation checklist. We hope this article was helpful! If you have any questions, don’t

hesitate to **contact us**.

**9**