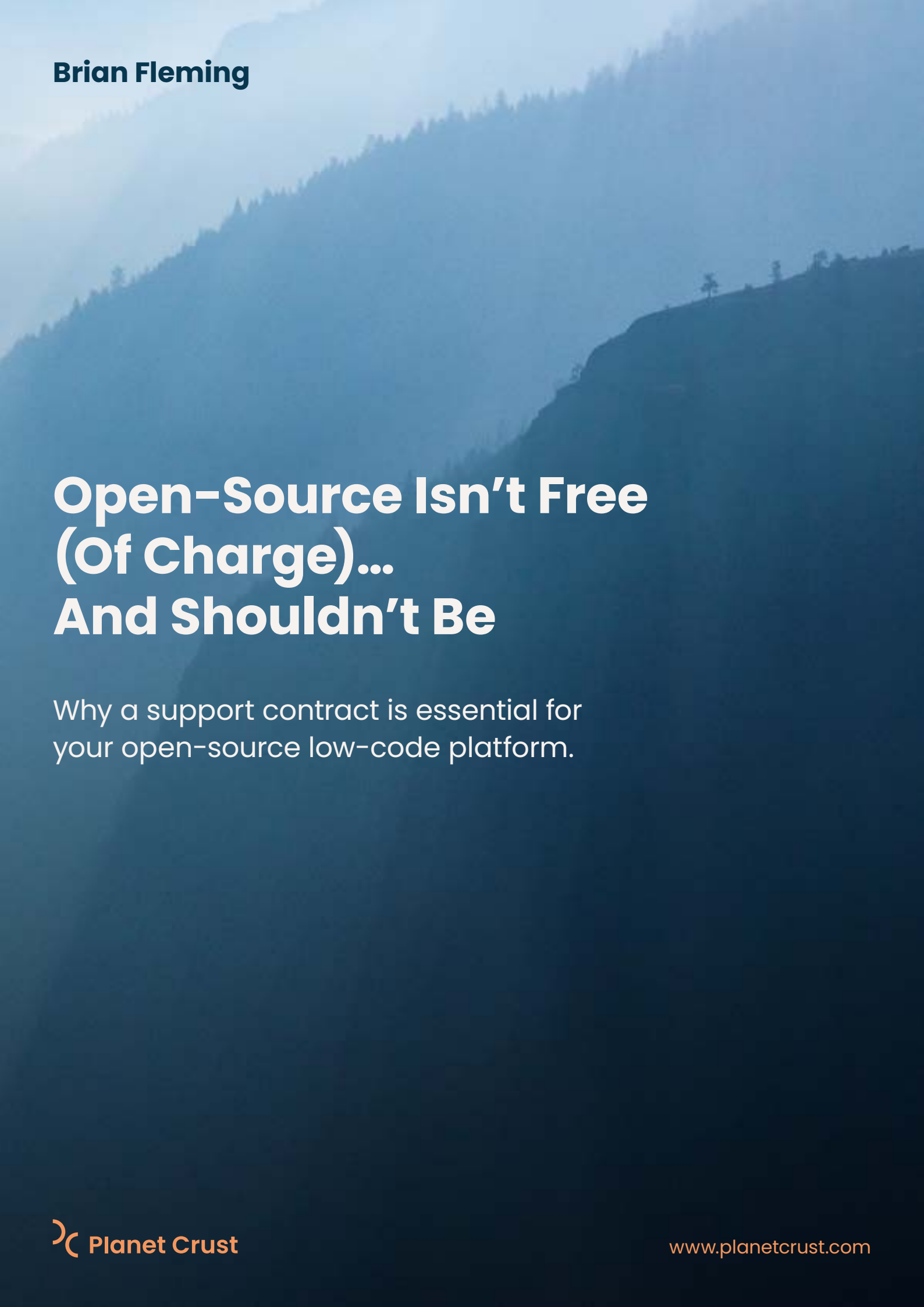
Brian Fleming



Open-Source Isn't Free (Of Charge)... And Shouldn't Be

Why a support contract is essential for your open-source low-code platform.

Planet Crust

www.planetcrust.com

Open-source software may be free to use, modify, and distribute. But that doesn't mean it's free of charge.

- Get an in-depth look at the open-source business model from both the customer and vendor perspective
- Learn about the open-source paradox and how it affects your organization
- Find out why you should always get a support contract, especially for open-source low-code platforms

"In the most common way of thinking, where 'free' means no upfront cost to use, modify, or distribute, the answer is yes: the software is free. That said, if you lean towards economics and like to think about the long-term costs of open source software, you may have brought to mind the old adage "There's no such thing as a free lunch".

– Karl Fogel
Partner at Open Tech Strategies, LLC

Table of contents

Introduction

When Finnish software engineer Linus Torvalds released the first version of the Linux kernel on September 17, 1991, few imagined it would end up marking a start of a revolution in the software industry. A month before releasing the new operating system, Torvalds casually remarked on a Usenet post that he was making a free operating system that wouldn't be big and professional like the GNU, and that it was just a hobby.

As it happened, Torvalds' hobby gave rise to the open-source software movement. Today, the internet as we know it couldn't exist without open-source software. Linux powers 96.3% of the world's top million web servers[1], and open-source content management systems (CMS) like WordPress and Joomla are behind millions of websites. Even Google Android, which powers around three-quarters of the world's mobile devices, is based on a custom Linux kernel.

Fast-forward to the 2020s, and it's clear that open-source is the undisputed future of software. 82% of IT leaders are now choosing to work with enterprise open-source vendors[2]. The move away from proprietary software continues to grow. Even more remarkably, the growth of open-source is being spearheaded partly by established software giants such as Microsoft, Amazon, and Google that had previously relied exclusively on proprietary software to drive revenue.

Few companies are more emblematic of the great shift in attitude towards open-source than Microsoft. The software giant, which once waged a legal battle against open-source, now uses open-source software itself. Its own cloud computing platform Azure runs largely on Linux-based data centers. Microsoft has even made its .NET source code open to the public, and it continues to improve compatibility with Windows via the Windows Subsystem for Linux.

Concurrent with this recent, broader embrace of open-source, we are now also seeing the rise of low-code and no-code app development platforms. These greatly broaden the reach of software development by saving developers time while also allowing business technologists to get involved in the development process. This can significantly reduce the workload on professional developers, while greatly reducing development times and costs.

Inevitably, there are already several open-source LCDPs on the market, including Corteza, WebCon, Budibase and Joget. This is no surprise, given that LCDPs are a natural fit for open-source. After all,

when you're developing apps in a low-code environment, it's better to have complete transparency into the underlying code in order to maintain complete digital sovereignty. With a closed-source LCDP, by contrast, you end up working in a black box that gives you limited control and ownership of the software you create and no view into what's going on inside it.

The benefits of adopting the open-source model for enterprise software are many, but it's also a fact that one of the most common reasons people choose open-source is because it's free. However, that doesn't necessarily mean that the total cost of ownership is nothing. In fact, without the right strategy, the TCO of open-source software can quickly grow out of control.

In this eBook, we'll explore the reasons for that and how to address them. In addition to looking at the open-source business model in general, we'll discuss:

- The true cost of using open-source software in your organization
- Why you should have a support contract for your open-source software
- How these matters concern the use of open-source LCDPs

The open-source paradox

At the heart of the open-source business model lies a central paradox that can most simply be summed up by the expression, *"You get what you pay for".*

To understand why this is so in the context of software development, and especially open-source software development, let's start by looking at what it takes to develop, support, and host open-source software or provide consulting services concerning it. We all know that it costs significant amounts of time, counted in years, and money, counted in the tens if not hundreds of thousands, to create any kind of software, never mind enterprise-level software, regardless of the licensing model it uses. Software development and engineering require a clear vision, detailed planning, and the work of highly skilled, and very expensive, employees with skills that have taken years to acquire.

And creating the software – usually in the form of a minimum viable product – is just the start. Once launched, the software needs to be supported, maintained, and continually improved in line with changing tech trends if it's going to stay relevant and useful to customers. Then there's also the matter of security. In this era of rapidly rising cyberattacks, software that isn't adequately maintained is a favorite target for ransomware, malicious code injection, and other threats.

To pay for the significant financial and human resources all this requires, software businesses need to simultaneously attract customers by letting them trial their software while also creating a recurring stream of revenue. In the closed-source realm, especially in the SaaS space, this is achieved either by offering a free trial of the complete software or by using a freemium model, whereby you offer basic functionality for free, then charge for the rest. In this way, if potential customers like the software or like the potential of its complete features, customers have no choice but to pay to get what they want. Added to this, if customers want to spend a few months "playing" with the software, they need to pay for this privilege, after which point they've become locked into the software.

It's this model that lets closed-source vendors lock-in not only customers, but also a recurring revenue stream that pays for improving their software and, consequently, continually pleasing their existing customers while attracting new ones. Done right, it's a continual feedback loop.

This is not so with open-source, where the software and all of its

underlying code are available for free. If someone wants to try the software, they can download it and install it on their own servers and get full functionality for as long as they want without having to pay anything. However, that will only get them the software they need now. But if the creators and developers of the software don't have a revenue stream, their business will go under; the software won't be supported, won't see improvements or innovation and will soon join the ash heap of outdated tech where it will become useless to the organization that enjoyed using it for free.

This is the open-source paradox. Yes, it's free, but unless customers pay to support it, it will soon be useless to them. At the same time, vendors give their software away for free, but unless they can convince people to pay to support it, it will soon wither and die. The problem is that many people think that "open-source" and "free-of-charge" are one and the same. Sure, the code is free to use, but all the rest – the support and maintenance, the regular improvements, the continual innovations – needs to be paid for, either through an organization's own, very expensive in-house development team or by entering into an assurance and support contract for the software.

It's about much more than getting something for free

The most commonly cited benefit of open-source software is that it doesn't cost anything to use. As we saw above, this is patently not true, because software needs to change over time. But even this is only a small part of the picture. In reality, open-source is radically changing the balance of power in the tech space to the benefit of a far broader range of individuals and organizations than software vendors alone.

On a geopolitical level, open-source even becomes a tool that entire nations can leverage to gain independence from the world's biggest tech giants, most of which are based in the US. With digital sovereignty being on the top of the mind in blocs like the EU[3], the crucial role of open-source cannot be understated.

However, on a company level, open-source means having total control and ownership of applications and data and much of the infrastructure behind them. As such, organizations can avoid vendor lock-in and all the restrictions that come with it. With interoperability and integrability being top priorities when facing increasingly disparate technology environments, that's a huge plus. To that end, open-source is more about *freedom* than about being free.

The true cost of maintaining open-source software

Some business leaders choose open-source software for the sole reason of reducing costs. In doing so, however, they miss the point that the open-source model offers many other key benefits, which they're highly unlikely to realize if they're only thinking about the free aspect. After all, it costs money to maintain the software, and the cost of doing so internally can easily grow out of control. This is one of the most common reasons for some businesses to choose proprietary software instead – they can see the results of the fees they're paying with each software release. Of course, if they stop liking the content of those releases and want to move to a different platform, they're stuck; as every business knows, migrating from one proprietary platform to another comes with a host of technical, business, and human resource challenges.

The true cost of maintaining open-source software in-house, on the other hand, can be difficult to quantify. While there are some easily quantifiable costs, such as hosting and computer power in the case of cloud-based solutions, others are hard to predict unless you have long experience as a software developer. For example, what do you do when you run into a problem with a piece of open-source software that a mission-critical system depends on? And how do you support, improve, and innovate the product over time to meet your changing needs? Doing so requires having a vision of exactly what those needs will be and which software you will need to meet them, i.e., a product roadmap. Dealing with these questions is difficult even for software vendors; you would, in effect, have to build your own internal software development department.

Even though people are free to download, use, modify, and distribute open-source software, the free aspect can be fast rendered irrelevant in the absence of an open-source budgeting strategy. Because of this, organizations must obtain a complete understanding of the total cost of ownership (TCO). This is vital regardless of the software licensing model concerned, yet it's also a common oversight in the case of open-source software. The TCO must include both the direct and indirect costs of the software throughout its entire lifecycle. While there may be other cost centers to consider, here are some of the most common:

- Freemium functions and features that may not share the same licensing model
- Additional development, testing, and deployment of the software
- Hosting, including storage and computer power and any bandwidth charges
- Support and maintenance costs, such as managed services or

break/fix contracts
- Indirect costs due to unscheduled downtime, such as lost revenue or productivity
- Ongoing training costs, such as bootcamps, workshops, and seminars
- Costs of planned downtime, such as during software upgrades
- Any direct or indirect liability costs

Most of the above are unavoidable, regardless of the licensing model. However, they can be much easier to quantify and predict for software that is supported by way of managed services. Businesses should also be wary of software, open-source or otherwise, that uses a freemium model whereby only certain layers of the software stack are open-source. Some vendors use both open and closed source models. For example, popular hypervisor technology VirtualBox is free and open-source, while its Extension Pack is premium proprietary software. While this isn't necessarily a bad thing, it's important to take any additional cost factors into account when working with a blended business model.

Managing risk is another key factor when determining the TCO for any kind of software. That said, open-source presents some unique risks. The main tradeoff in open-source software is that it's provided for free without any guarantees. While this doesn't necessarily mean that using open-source software inherently carries more risk, it does mean that, in the absence of a third-party support and maintenance contract, the user must assume all of the risk. The risks can be offset considerably in the case of open-source software that has a large and active community behind it but, again, there are no guarantees.

When choosing open-source software, decision-makers must factor in the risks involved. The best way to manage these risks is to break them down into categories, such as security, legal, operational, business, and reputational. Once the risks have been identified, the next step is to factor in what they mean to your business before figuring out how to mitigate them. This is ideally achieved using a shared responsibility model, which we will discuss later.

The issue of bugs

Let's face it, no software is ever released that doesn't have bugs, hence the need for updates and patches. And yet there is often a perception that proprietary or closed-source code is higher quality and has fewer bugs compared to open-source code.

This could be for a number of reasons, including the very nature of open-source versus closed-source software, i.e., the bugs in open-source software are as freely available to see and find as the software itself, whereas with closed-source the opposite is the case; if you can't look inside the source code, you can't see the bugs. Or it could be a function of marketing – it's no stretch to say that closed-source software vendors typically have slicker marketing and tighter messaging and control over information than open-source software vendors typically do.

Whatever the reason for this perception, it's false. The reality is that open-source software is no more and no less buggy or secure than closed-source. Bugs are simply a fact of software life. Whether those bugs are found really comes down to the dedication of the people and teams searching for the bugs, regardless of whether they're working in the open-source or closed-source domain.

So, how does this relate to you as a software user?

Firstly, with open-source, you always have the option of digging into the source code yourself to find and fix any bugs. But for reasons we'll discuss below, this isn't the most efficient nor cost-effective approach. A smarter, approach is to rely on a support service provider to prioritize and fix bugs that are critical to you. This is a level of service that goes far beyond simply supporting, maintaining, and improving the code and instead focuses on supporting, maintaining and improving the code so it suits *you*.

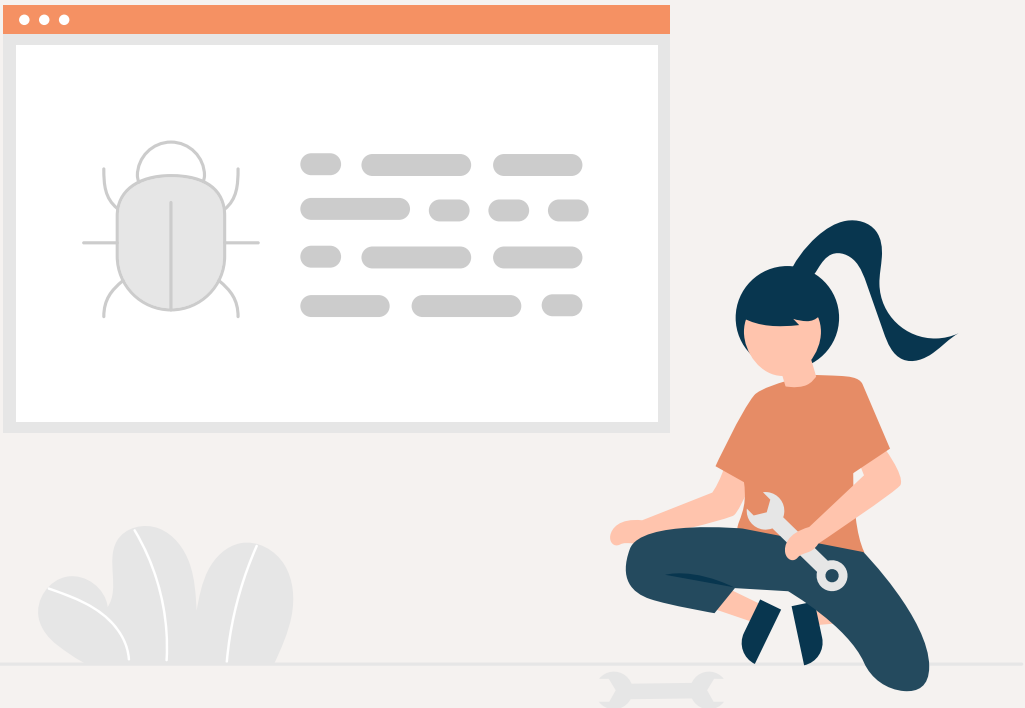
To understand why they would do this, let's return to the open-source paradox.

With closed-source software, the cost of general support for and maintenance of the software, including bug hunting and fixing, is baked into the price you pay for it. With open-source, it's not – the software is free. Bug hunting and fixing is taken up by the general community, if it's large enough, or by dedicated developers and maintainers of the software, be they the original creators or third parties. These developers and maintainers naturally need to secure a revenue stream to fund their bug fixing, which they do by offering support for the software. This leads to a situation where, if every

user of open-source software pays for a support contract, everyone benefits from improved software, while if no one pays, everyone loses. Moreover, the software stagnates and, ultimately, fossilizes. Inevitably, there is always someone who thinks, "*I don't need to pay because someone else will.*" But it's easy to see where this logic will soon lead – if everyone thinks this way, the software will soon die.

To avoid this happening, open-source vendors often look for ways to sweeten the pot to attract customers that go far beyond what closed-source vendors typically include in their support. For example, closed-source vendors won't necessarily address or even care about those critical bugs you care about the most that stand in the way of you doing what you need to do. This is particularly grieving when you're using an LCDP, where a bug may prevent you from completing an app, workflow, or integration that's critical to your operations and stopping your efforts in their tracks with no way to move forward. At best, a closed-source vendor may put your bug-fix request on a list of other fixes with no particular priority, or charge you an arm and a leg for that level of support.

Open-source vendors, on the other hand, are incentivized by the open-source paradox to offer you prioritized bug fixing, along with a host of other service offerings that we'll look at in a bit. The end result is not only better-quality code, but better-quality software that will stay with you and improve and innovate over the long run. But that can only happen if people pay.



Ensuring your platform does everything you need it to do

Open-source software is often still viewed as having a significantly steeper learning curve than closed-source software. For example, Linux-based operating systems have a reputation of being for tech professionals and enthusiasts. However, this perception is outdated, particularly in the case of larger-scale open-source projects that have large communities supporting them.

That being said, realizing the true potential of open-source software in terms of the flexibility it offers is often quite a different matter. For example, anyone with moderate technical skills can design an app using a platform like Appian or Mendix, but they will only be able to do so within the confines of those environments, unless they can change the underlying code. Similarly, anyone might be able to use the popular Linux distro Ubuntu, but it's impossible to significantly alter its functionality without coding.

The same applies to any open-source software, and it doesn't stop at coding either. There is user experience design and third-party integrations to think about, and much more. To ensure that an open-source platform really does everything you want it to do, you will need extra help, either in the form of a well-resourced in-house team or an outsourced consulting and managed services firm. The more unique and complex your requirements, the more important that is.

Software is, of course, a vital part of the value stream in any modern business. This is why it's essential to start with a detailed analysis of your business requirements before matching those requirements to the appropriate solution. Open-source offers another fundamental advantage here: the open nature of the software means having far greater flexibility to tailor the platform to a unique and broad range of needs. By contrast, proprietary software tends to be designed only for a select few specific use cases and, while it might address those use cases very well, there are few or no opportunities for further innovation.

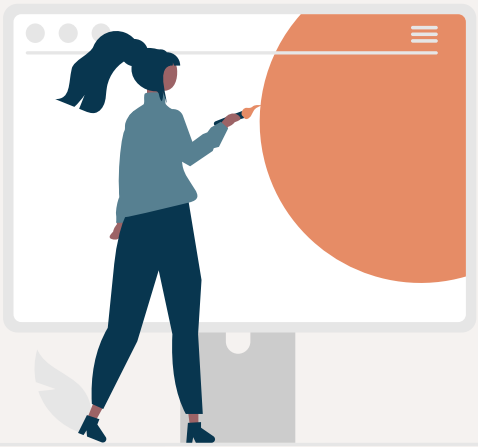
Partnering with a consulting firm when deploying and maintaining open-source software offers a dependable way to reduce and gain control over many of the direct and indirect costs that we mentioned earlier. Most importantly, consulting services will help you determine how the software in question might be tailored to meet the precise needs of your business. Moreover, if you feel that the platform is missing an important feature, you can simply request that it be developed for you.

Even more importantly, consulting services can greatly reduce the risks associated with open-source software. By helping all stakeholders get a closer understanding of how the platform would integrate into their broader technology environments, they will be better prepared to patch potential security holes, customize user interfaces, and ensure integration with legacy software and third-party services. With the right team to help you get the most out of your open-source software, you can deploy it at less risk, with greater visibility into implementation costs, and ensure that your platform is as future-proof as possible.

Finally, teaming up with a consultant can help you mitigate risks associated with intellectual property rights. While commercial software tends to come with very clear licensing rules, the same cannot be said for a lot of open-source software. Often, the licensing is ambiguous. For example, the popular GNU General Public License requires the free, unrestricted distribution of the source code of a modified version of software, even if that source code is incorporated into proprietary software. This is known informally as copyleft, which can have implications for companies. On the other hand, an Apache 2.0 license comes with no such copyleft requirements. In fact, the Apache 2.0 License lets you:

- Use the code or any part of it in your commercial products for free
- Use, modify, distribute, or sell the code or any part of it as you like
- Release your modified version of the code under any license of your choice
- Claim copyright over your configuration files and issue binaries as proprietary software

Furthermore, all the above comes with no obligation to publish your modified code.



Reducing the burden of software maintenance and support

No mission-critical software application can be left unsupported. Software maintenance costs typically account for around 75% of the TCO[4]. This spans corrective, adaptive, and perfective maintenance, in which software is repaired, modified, or enhanced respectively. These costs are often hard to budget for too. After all, you never know when you might need to patch a bug or introduce new functionality until the last minute. When that happens, the costs can quickly rack up.

Traditionally, when maintaining any kind of software, organizations have had to rely entirely on the break/fix support model, a standard fee-for-service method in the software industry. The problem with this model, while unavoidable in certain scenarios, is that it's often difficult to determine how much a fix or modification will end up costing. In the case of proprietary software, there's also the issue that only an internal team may have the knowledge and experience required to maintain the software. If such expertise is no longer available in the organization, there will be no one to effectively maintain it. Off-the-shelf software, on the other hand, is typically supported by the vendor but, as we discussed above, things like bug fixes are often handled at the vendor's discretion or cost the customer a premium.

Open-source software, however, comes with multiple support options. The first option is often to ask the community for a fix, but there's no guarantee that the response will be timely enough or even correct. Most open-source communities have extensive knowledgebases and forums, where it's typically easy to find solutions to recurring issues. If, however, you're encountering an issue that hasn't been addressed before, then it can take much longer to find a solution. For projects supported by very large communities, this problem may be significantly offset, but it will still be a factor. If we're talking about mission-critical business software that you simply can't function without, then that's a clear dealbreaker.

Of course, another option might be to address the issue yourself. However, that requires a suitably staffed internal team with the resources necessary to solve the problem or implement a new function within a given timespan. For smaller organizations, which typically don't have a fully-staffed team of professional software developers and engineers, that's not an option. This may be different for large enterprises, but even so, they don't have the expert knowledge of and experience with the source code that the developers of the software have. And while they could invest the

time and effort to gain that knowledge and experience, they would likely have to hire additional developers to do so while staying on top of their current workload.

The most practical option for most organizations embracing open-source is to have a support contract with a service provider, ideally, one who is the creator of or at the very least a major contributor to the project and knows it inside out. With enterprise-grade support, you're protected with a service level agreement (SLA) that guarantees response and resolution times, as well as other areas of maintenance, such as patching and feature updates. By contrast, community-based support comes with no SLAs and no guarantees, and there's no single, 24/7 point of contact for when something goes wrong either.

The best things about having a support contract is that: 1) it provides complete visibility into costs, without the surprise extra costs that come with relying on the break/fix model; and 2) knowing that your breaks will be resolved quickly, so you can get on with your work and that your bugs will be prioritized. These two factors make it much easier to determine the TCO and budget for the longer term. The risks are also greatly mitigated thanks to a shared responsibility model, whereby the service provider assumes most of the risk. Furthermore, many vendors, including Planet Crust, offer multiple support tiers, allowing you to choose an option that best suits your specific requirements and operational environment.

What does this mean for open-source low code platforms?

Low code software development platforms (LCDP) greatly reduce development time and ease the burden of software maintenance by abstracting much of the software development away from the underlying code. This is typically achieved using intuitive drag-and-drop interfaces that allow end-users to quickly implement common software functions and interface elements without having to resort to coding.

While there are some open-source LCDPs such as Corteza, which is developed and maintained by the Planet Crust team, WebCon, Budibase and Joget, most LCDPs are closed source. For closed-source LCDPs, you're not only limited in terms of design flexibility, but also with regards to how you can use and distribute any apps you build with the platform. For example, many LCDPs only allow you to sell apps you make on their own marketplaces, while others might not even allow you to sell them at all.

Open-source and low code are ideally matched, since the open-source element ensures that you have complete digital sovereignty and ownership rights. The low code factor ensures you get to enjoy many of the advantages of rapid application development (RAD) in a way that's accessible to a much wider audience beyond professional software developers alone.

Naturally, it might sound counterintuitive to pay anything at all to use an open-source LCDP. After all, if the license is open-source and the platform is far more accessible than a traditional software development environment, then why would you need to pay? The answer to that is, of course, that you don't need to pay, but there are some clear benefits to taking out a support contract, including:

- Accessing prompt, enterprise-grade support whenever you need it
- Consolidating hosting, storage, and support into a single bill
- Understanding your intellectual property requirements and obligations
- Collaborating directly with and supporting the open-source community
- Reducing the risks associated with the break/fix support model
- Maintaining control over and visibility into your total cost of ownership

Beyond this, premium-level support contracts will often allow you to collaborate directly with developers, provide help with white labelling, and give you guidance on your intellectual

property rights. They will typically also give you access to the core development team, business analysts, training, and the possibility of co-developing parts of the software. Furthermore, with a high-level support contract, you'll usually have a say in the software's roadmap by being able to request features that solve the problems you need solved. This ensures that the software develops in a direction that matches the direction your organization is going.

So, yes, open-source software is, by definition, distributed free-of-charge. But focusing on the free element misses the point that all software costs time and money to develop, and much of the costs involved are tied to the ongoing maintenance of the software. Maintaining software in-house can quickly end up being far more expensive than having a support contract, which gives you the services you need for a set monthly fee. In the case of LCDPs, having a support contract means you can get much more out of the platform while reducing risk across security, business, legal, and operational domains.

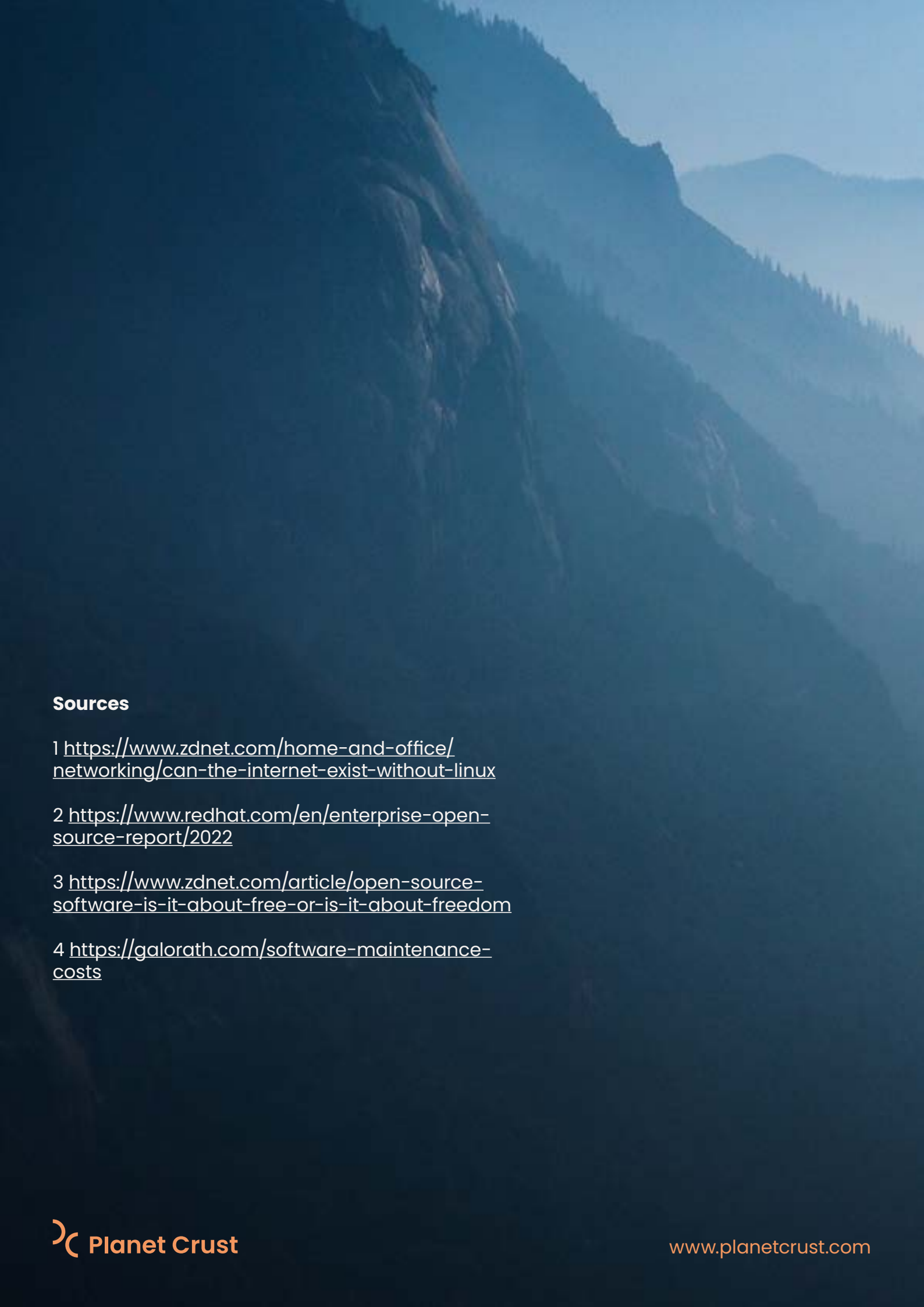
On top of this, your support contract is helping not only to future-proof the software, but to future-proof its utility to you. Closed-source software can't promise the same at such a granular level, whereas using open-source software without a support contract that contributes to its development leaves you at the mercy of whatever direction the supporters, maintainers, and developers of the software want to take it in.

Conclusion: A better way to pay for open-source software

In the end, having a support contract for your open-source software is about having a sense of responsibility, particularly in the case of mission-critical apps in a production environment. If your business depends on an LCDP for essential functionality for your operations, it makes sense from any perspective to invest in a support contract. It provides that vital extra layer of protection and allows you to enjoy all the benefits of both open-source and low code, such as the ability to innovate at scale with complete agility.

Looking at the bigger picture, paying for support services supports the further development of open-source software. For example, Linux would not exist if it didn't have its contributors, and the more influential among those contributors rarely have the inclination nor the resources to work for free. That's precisely why major for-profit companies like Microsoft, Red Hat, and Google are all embracing the Linux-based kernel to grow their own revenues and contribute to the broader project. Furthermore, customers also have a say, since they can request and even co-develop new features.

Planet Crust *helps independent software vendors, enterprises, and public sector organizations power and optimize their business processes at a fraction of the cost of traditional providers. Our team is the primary contributor to the* ***Corteza*** *digital work solution, a low-code platform that facilitates the development of scalable cloud apps and enterprise-grade CRM systems. We also offer consulting, customization, and support services to help you get the most out of the platform.* ***Get started for free*** *today.*



Sources

1 https://www.zdnet.com/home-and-office/networking/can-the-internet-exist-without-linux

2 https://www.redhat.com/en/enterprise-open-source-report/2022

3 https://www.zdnet.com/article/open-source-software-is-it-about-free-or-is-it-about-freedom

4 https://galorath.com/software-maintenance-costs