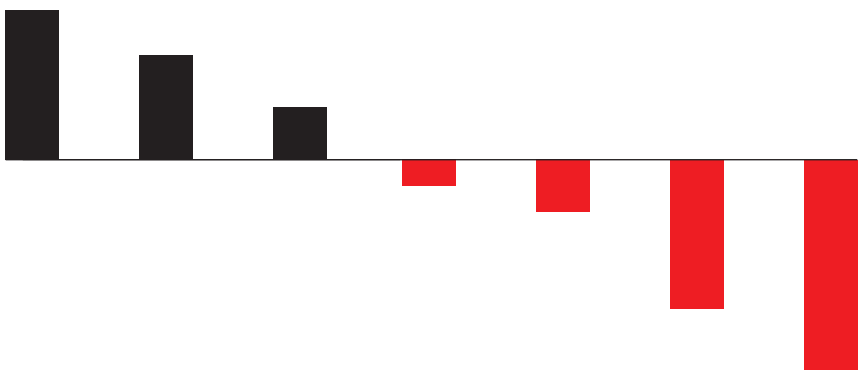
O'REILLY®

THE SOFTWARE PARADOX

The Rise and Fall of the Commercial Software Market



Stephen O'Grady

The Software Paradox

Stephen O'Grady

O'REILLY® Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

THE SOFTWARE PARADOX

by Stephen O'Grady

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

March 2015: First Edition

Revision History for the First Edition:

2015-05-18: First release

See *http://oreilly.com/catalog/errata.csp?isbn=9781491900932* for release details.

ISBN: 978-1-491-90093-2
[LSI]

Contents

What Is the Software Paradox?

par·a·dox

ˈparəˌdäks/

NOUN: paradox; PLURAL NOUN: paradoxes.

> *A statement or proposition that, despite sound (or apparently sound) reasoning from acceptable premises, leads to a conclusion that seems senseless, logically unacceptable, or self-contradictory.*

On Wednesday, August 12, 1981, IBM introduced the Model 5150, which the world would come to know as the Personal Computer (PC). The base price for a version without disk drives was $1,565, or just over $4,000 in today's dollars after adjusting for inflation. While it was launched with much fanfare and would become the foundation for a revolution in hardware, the PC was not the first of its kind to market. Steve Jobs, Steve Wozniak, and Ronald Wayne had introduced the Apple I, in fact, five years earlier in July of 1976. The Apple II followed in 1977, the same year that Commodore's PET 2001 was announced at the Consumer Electronics Show.

Though its focus had historically been on technology for large businesses, the PC market, which transcended enterprise and consumer markets, was for IBM, both opportunity and threat. The argument can be made, in fact, that the 5150 was rushed to production, a hasty response to a market whose potential IBM had substantially underestimated. Certainly it represented a departure from the Armonk giant's historical design process, in which IBM hardware was built using components designed and built by IBM. With demand for personal computing exploding, the company resorted to outsourcing. Unlike its traditional mainframe hardware, the PC was built instead from available off-the-shelf components sourced from external suppliers. Instead of incorporating the IBM 801 processor, for example, the PC relied on the less powerful Intel 8088 chip. By optimizing for components

that could be efficiently sourced, the product's time to market was greatly accelerated: the 5150 was designed in about a year.

With startups like Apple growing quickly and large existing vendors like IBM validating the market, the age of the PC was at hand. As *Time* Magazine acknowledged, in 1982, its Person of the Year was not a person, but "The Computer."

In retrospect, the most interesting aspect to the launch of the PC was how unimportant the software appeared to be. Following one of journalism's cardinal laws, most of the attention followed the money, which led inevitably to hardware. Commercial software businesses existed, to be sure—Oracle, for example, was four years old when the PC was launched—but software at the time was viewed as more of an enabler for hardware than a standalone market. When the PC debuted, hardware-centric IBM was worth almost 34 billion dollars; neither of the software-based duo of Microsoft and Oracle would even be publicly traded for another five years.

As a result, the software powering the PC was something of an afterthought. Viewing the operating system software that would serve as the foundation for its new platform as even less strategically important than its hardware components, IBM was content to contract the development of the software to a third party. After failing to come to terms with Gary Kildall of Digital Research, they turned to a small company called Microsoft. Microsoft, in turn, purchased the basis for their PC operating system from yet another third party, Tim Paterson of Seattle Computer Products. In the end, Microsoft's MS-DOS operating system, rebranded as PC-DOS on the IBM PC, became the default operating system for a new wave of hardware, shipped in volumes without precedent.

For the small company that Microsoft was at the time, a distribution deal with a behemoth like IBM would have been, by itself, akin to a winning lottery ticket. But like his contemporary from another industry, Bill Gates had a much bigger prize in mind.

When George Lucas was negotiating with 20th Century Fox prior to the filming of the original *Star Wars* film, he had the option to negotiate for more upfront compensation. His 1973 film *American Graffiti* had been an unexpected success, and highly profitable for the studio. Instead of using this leverage to maximize his upfront capital return, however, he instead obtained from the studio control of the final cut, 40% of the box office gross, and most important, merchandising rights associated with the franchise. In a deal that will never be repeated in Hollywood, George Lucas left a few hundred thousand dollars on the table in his contract in exchange for hundreds of millions of dollars of future income.

Just as 20th Century Fox dramatically underestimated the value of those rights, so too did IBM fail to comprehend the importance of the software operating system. Gates, however, had uniquely perceived the revenue opportunity in software as a standalone entity when he and Paul Allen had been building BASIC compilers for various operating systems in the late 1970s. In what would later look like a heist, he was able to extract from IBM the contractual ability to license and sell MS-DOS outside the 5150 product. While this looks like a foolish mistake in retrospect, it is less surprising if you consider the context of the time, which was a market that attached little commercial value to software as an asset. IBM was unable on a fundamental level to comprehend the commercial opportunities that software represented, because it shared the wider market's opinion that the money was in hardware, not software.

Five years after the release of the IBM 5150, Microsoft went public. On March 31, 1986, the company was worth $679 million. On that same date, IBM was worth $93 billion.

Fewer than 10 years later, Microsoft—the one-time David to IBM's Goliath—was worth more than IBM. The bulk of this valuation, of course, was fueled by software—specifically Office and Windows. At its peak on December 27, 1999, in fact, Microsoft was worth $613 billion dollars, or a little more than three times what its one-time partner IBM was worth at that time.

Software, it seems, had some commercial value after all.

The past few decades have, in general, been good ones for software. Once an afterthought, software became not just a means to an end but an end in and of itself. Trillions of dollars of wealth were created by software vendors and the markets they created and owned. The ascension of software was perhaps best described in a now-famous *Wall Street Journal* op-ed by Marc Andreessen on August 20, 2011, "Why Software Is Eating the World." In the piece, the man whose fortune was made in part by the $2.1 billion IPO of the software company Netscape described the present state as the following:

> *More and more major businesses and industries are being run on software and delivered as online services—from movies to agriculture to national defense. Many of the winners are Silicon Valley-style entrepreneurial technology companies that are invading and overturning established industry structures. Over the next 10 years, I expect many more industries to be disrupted by software, with new world-beating Silicon Valley companies doing the disruption in more cases than not.*
>
> **— MARC ANDREESSEN**

By the time Andreessen wrote those words, there were few who would disagree with the core thesis. Those who would were most likely to be employed by industries in the process of being actively disrupted by software. Software was, and still is, the new reality for most industries. Much as Amazon is now more appropriately described as a technology company than a retailer, so too are an increasing number of businesses in an ever-widening number of industries.

A curious thing was happening while software was hungrily consuming the world, however. Even as it was becoming more and more vital and disruptive, software's commercial value was declining. Software that would have once generated billions in revenue per quarter is increasingly made available for free. Companies that once battled each other and struggled to differentiate similar proprietary products now collaborate with each other on a common platform, competing on implementations and service. Developers that solve interesting problems with software see more benefit than cost to making it available for free than attempting to charge for it.

This is the Software Paradox: the most powerful disruptor we have ever seen and the creator of multibillion-dollar net new markets is being commercially devalued, daily. Just as the technology industry was firmly convinced in 1981 that the money was in hardware, not software, the industry today is largely built on the assumption that the real revenue is in software. The evidence, however, suggests that software is less valuable—in the commercial sense—than many are aware, and becoming less so by the day. And that trend is, in all likelihood, not reversible. The question facing an entire industry, then, is what next?

This is the question the following pages intend to answer.

The Evidence?

The weight of evidence for an extraordinary claim must be proportioned to its strangeness.

— MARQUIS DE LAPLACE

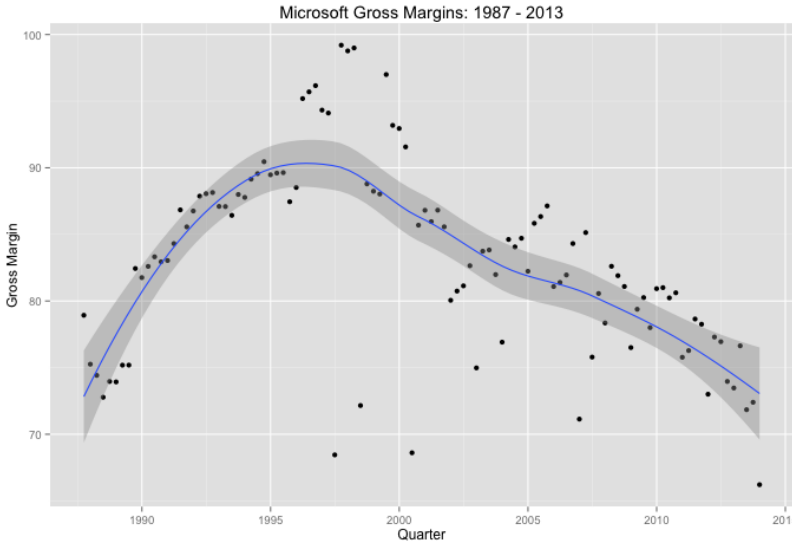
Given the returns that the commercial software industry has generated historically and is still generating today, the typical reaction to the hypothesis that realizable commercial values of software as a standalone entity are in decline is skepticism. Which is entirely appropriate, given the extraordinary nature of the claim.

In 2013, Microsoft's Windows and Office (Business) divisions collectively generated $44 billion in revenue, up 4% from 2012, which was in turn up 3% from 2011. In one year, then, Microsoft generated more from two software products than VMware, Yahoo!, Salesforce, Adobe, Twitter, Nokia, Netflix, or Intuit are worth as companies. How then does one construct the argument that it's becoming more difficult to sell software, at Microsoft or more broadly?

With Microsoft, it's surprisingly simple. It is true that Microsoft continues to excel at generating software revenue. Even if we allow that this is largely an artifact of that rarest of achievements—a true monopoly—the company's ability to maintain its dominance over decades despite fierce competition and an industry that is always in change around it proves one thing: Microsoft can make money with software. The question with Microsoft, therefore, isn't whether they can make money, it's whether they can make money as efficiently as they have in the past. Because if one looks beneath the surface of their financials, there appear to be cracks in the façade.

Microsoft's ability to generate revenue remains unchallenged, but their ability to extract profit from that revenue has proven more difficult to sustain. In the third quarter of 1987, one year after going public, Microsoft posted a quarterly profit margin of 79%. At its peak in 1999, Microsoft would post an average profit margin of 93%. Since the first quarter of the year 2000, they have never again broken the 90s. Microsoft's margin in the last quarter of 2013, meanwhile, was its worst ever

at 66%. The following chart depicts Microsoft's gross margin over time (with a line of best fit and confidence interval).



For Microsoft and its shareholders, the trajectory implied here is troubling. Microsoft retains an unparalleled ability to generate revenue from software, but given that it's able to generate less profit from that revenue today than it did a year after going public, it seems important to question the mechanics of its model moving forward. As the company seems to be doing: it's no accident that Steve Ballmer's replacement, Satya Nadella, came from Microsoft's cloud team. Nor that the company is releasing free versions of its operating system and office suite for mobile, or that it was willing to risk partner relationships and commit massive financial resources to enter hardware markets in both cloud (Azure) and mobile (Surface). It may or may not be an accident that Bill Gates is currently on track to have no direct ownership of Microsoft in four years.

Other software-first industry players are in the midst of their own transitions. For the first time in almost five years, German software provider SAP initiated widespread job cuts over the summer of 2014. Their purpose? Jim Dever, a spokesman for the company, told Bloomberg that the "company is eliminating jobs across divisions as it seeks to move faster and deliver more of its products as online cloud-computing services instead of software that runs in customers' data centers." As with Microsoft, which built itself on the sales of on-premise software, SAP is compelled by the market to change the very nature of its business.

But is the Software Paradox truly a systemic, industry-wide issue, or is it better characterized as mere failures of execution? To explore this question, we need to broaden our scope. In May 2013, the consultancy PwC compiled a list of the Top 100 companies in the world as measured by software revenue, the Global 100 Software Leaders. Here are the top 25:

1. Microsoft
2. IBM
3. Oracle
4. SAP
5. Ericsson
6. Symantec
7. HP
8. EMC
9. Computer Associates
10. Adobe
11. VMware
12. Fujitsu
13. SAS
14. Intuit
15. Siemens
16. Dassault Systemes
17. Autodesk
18. Salesforce
19. BMC
20. Hitachi
21. Infor
22. Sage
23. Cisco
24. Intel

25. Citrix

If you examine the companies that make up that list, one commonality that leaps out is their age. The average Top 25 software company, by PwC's metrics at least, is around 46 years old. But that's admittedly skewed by outliers such as IBM, which is 103 years old, or Siemens, which is somewhat incredibly 167. The skew-resistant median, however, is 34 years of age, which means that the average large software company was founded the year that John Lennon was shot and killed, the year Tim Berners-Lee began work on the system that would lead to the World Wide Web, and the year that Lucas' *The Empire Strikes Back* hit theaters.

The fact that the largest companies in a given industry are some of its oldest is, to be sure, hardly unusual. Growth through acquisition is a common pattern in most industries, and is particularly popular in technology. This is even more common in markets where high and low margin opportunities exist; the former tend to use their economic advantages to "compete" against the latter by purchasing them. From a pure development model perspective, larger technology vendors have long outsourced research and development to startups, believing that the cost of the acquisition premium is more than offset by lowered risk and costs with better product predictability. Paul Graham described this process well in a 2005 essay entitled "Hiring is Obsolete."

> *Big companies also lose because they usually only build one of each thing. When you only have one Web browser, you can't do anything really risky with it. If ten different startups design ten different Web browsers and you take the best, you'll probably get something better.*
>
> *The more general version of this problem is that there are too many new ideas for companies to explore them all. There might be 500 startups right now who think they're making something Microsoft might buy. Even Microsoft probably couldn't manage 500 development projects in-house.*
>
> **— PAUL GRAHAM**

But in a such a dynamic industry, the age of its largest entities is still something of a surprise. It should be difficult to build a long-lived software company, given the engineering preference for new problems over old ones.

Technology businesses have tended to be more vulnerable to disruption than their counterparts from other industries, as sudden advances in technology within

or adjacent to a particular market can obsolete a given business's products almost overnight. While GM has not had to worry, for the most part, about the automobile being replaced by an alternative means of transportation, technology industry players have had to weather tectonic shifts from mainframes to mini-computers, PCs to mobile, servers to cloud, and so on. Technology disruption is in part what has permitted Facebook, Google, LinkedIn, and Twitter to generate over $600 billion in collective market value in 16 years, or just about half the time the average PwC software vendor has been in existence. It's important to note that none of Facebook, Google, et al., are included on PwC's list, however, due to the simple fact that none of them happen to sell software directly.

Of the large, mature organizations with large software revenue streams on PwC's Top 25, just how critical is software to their overall balance sheet? Is it their primary income stream, or merely one of multiple large sources of revenue? One would assume that because these are the 25 largest companies in the world as measured by software revenue, software would be the dominant business model among the majority of the members of the list. As in fact it is, if only a slight majority. Of PwC's Top 25 software players, 15 (or 60%) derive the majority of their income from distributed software sales. Another way of stating that, however, is that of the 25 largest software vendors in the world, almost half do not make the majority of their money from software.

But what about the companies not on PwC's list? If we acknowledge that the importance of software as a revenue stream is something less than dominant within the largest software earners in the world, the next logical question is what role software plays within the technology market as a whole. What if, for example, the scope was expanded yet again, this time beyond PwC's strict subset of software-oriented vendors? If we looked for the largest "technology companies" rather than the largest "software companies," for example, we would be able to add large players like Apple or Google. If we then sorted by market capitalization rather than estimated software revenues, that list might look something like this:

1. Apple
2. Google
3. Microsoft
4. Samsung (Consumer)
5. Verizon
6. IBM

7. Oracle
8. Facebook
9. AT&T
10. Amazon
11. Samsung (LCD)
12. Qualcomm
13. Intel
14. Cisco
15. Siemens
16. SAP
17. Taiwanese Semiconductor
18. Baidu
19. HP
20. EMC
21. Texas Instruments
22. VMware
23. Ericsson
24. Yahoo!
25. Salesforce

This list is even more interesting with respect to the role of software. Of the 25 largest technology companies in the world on this list, 21 (or 84%) derive the majority of their revenue from something other than traditional software licensing and sales (i.e., not hardware, SaaS, or services). Broadly speaking, then, it seems clear that as important as software is to the technology industry—and make no mistake, it is fundamentally crucial—it is directly responsible for a distinct minority of the revenue, at least when sold as a standalone product.

If the macro evidence is suggestive, what about the micropicture? What about, for example, the software products themselves? Is there any evidence that the Software Paradox is manifesting itself directly within current product pricing? In nearly all industry categories, the answer to this question is yes. Consider the PC operating system market. In March of 2001, Apple debuted OS X 10.0, the first major release

of its current desktop operating system. The retail cost for the product at the time was $129, or around $173 in 2014 dollars. A decade later in 2011, version 10.7, codenamed Lion, was made available via Apple's Mac App Store for $29.99. Two years after that, 10.9, also referred to as "Mavericks," was released via the same channel at no cost.

It seems reasonable to assume that the cost of production for OS X did not suddenly drop to zero over the course of 12 years, which implies that this is a statement from Apple about the commercial value of the software. Specifically, that the company no longer felt that the operating system was monetizable. Even Microsoft, whose market capitalization was built in part on the back of operating system licensing fees, is said to be planning a free version of same.

Nor is the decline in realizable PC operating system revenue simply a consequence of the decline in importance of that market. If this were true, the category making the biggest gains at the PC market's expense, mobile, would be expected to be a major new source of operating system licensing revenue. We would simply see a wealth transfer between PC operating system players to mobile operating system providers. Instead, the availability of Android source code and the success of Apple's integrated hardware and software strategy has made it difficult if not impossible for vendors to replicate the retail operating system model in mobile. Microsoft has had some success generating a licensing-like revenue stream by virtue of intellectual property and patent licensing, but the viability and growth potential of that approach longer term is questionable. As for licensing of its own operating system, Microsoft has acknowledged that the market value for that is zero: it announced in April of 2014 that for devices with a screen size of 9 inches or less, its mobile operating system would be available at no cost. As explosive as the growth in mobile has been then, software licensing revenue has not been a major beneficiary. Even for mobile apps, the revenue opportunities have been limited. As Instapaper creator Marco Arment said in 2013, "Paid-up-front iOS apps had a great run, but it's over. Time to make other plans."

Whether the lens used is market conditions, or the performance of bellwether software entities, or even individual products, the trend is the same: it is growing more difficult to sell software up front, on a standalone basis. More important, however, the market appears to be pricing this into its valuations, favoring models that make money with software over those attempting to make money from the sales of software.

The Four Generations of Software Valuation

While this sustained decline in what has been a lucrative market for multiple decades might come as a surprise to many, the truth is that this is merely the return arc of a pendulum swing, one in which the price of software has swung wildly in one direction and now is on a return path. Consider the following generational attitudes toward software:

First Generation (1950–1986)

Best characterized by IBM, this type of technology provider firmly believed that software was a means to an end rather than an end in and of itself. Which, given the difficulties and expense associated with manufacturing physical hardware, was understandable. The SHARE user group founded in 1955 by IBM 701 users was one of the manifestations of this attitude; one of its major resources was its library, which consisted of patches to the operating system that were possible only because IBM made the source code for its operating system available to users. Why? Because the money was in the hardware, not the software, and anything that would improve the company's ability to sell its hardware, such as software optimized by the users themselves, was perfectly logical. It was not until 1968, in fact, and only under pressure from the US government, that IBM began to charge separately for its software. This strategy left a variety of players vulnerable to the succeeding generation's software monetization efforts.

Second Generation (1986–1998)

While IBM's prior experience with hardware and operating systems had led it to conclude that the money was in the former rather than the latter, Microsoft's unique realization was that the reverse might in fact be true. Believing that software represented a classic under-appreciated asset, Microsoft seized on this opportunity and built itself into one of the largest companies in history, almost strictly through revenue generated from the sales of the software it created. Just as IBM's experiences led it to believe that the real revenue opportunity lay in hardware, however, Microsoft's dramatic software-fueled growth led it to conclude that software was the once and future revenue opportunity, which opened the door to the next generation of provider, who again had differing ideas on the value of software in economic terms.

Third Generation (1998–2004)

With Microsoft absorbing a disproportionate share of revenues and well placed strategically and financially to respond to competitive threats in the area of

software, a new class of technology provider emerged that was built off of software, but in a fundamentally different way. By engaging directly with users via a browser, Google was able to effectively bypass Microsoft's dominant positions in various software markets and build itself into one of the largest technology companies in the world today—all without selling so much as a single license of distributed software. Core to its success was the realization that the economics of scaling itself to a worldwide audience using proprietary software were untenable, which led the company to build itself upon open source software. This ability to construct a massive, global technical infrastructure using little-to-no proprietary software naturally led to questions about what software was actually worth. Cowen & Co. analyst Peter Goldmacher estimated in 2011, as an example, that Google-owned YouTube would have spent nearly six times as much building out its infrastructure on Oracle Exadata versus open source software and commodity hardware alternatives. But while Google was not built upon a model of monetizing software directly, as was Microsoft before it, its behavior suggests that the firm does believe software can still be differentiating. Instead of directly open sourcing pieces of its infrastructure like Dremel, Pregel, or Spanner, Google instead publishes publicly the details required to implement them, giving the community the opportunity to implement their own version, as it did with Hadoop following the Google Filesystem and MapReduce papers.

Amazon, though founded four years before Google in 1994, shares the search provider's core philosophies in terms of the importance of open source code and the need to protect its own innovations. Amazon.com and its AWS subsidiary are voracious consumers of open source code, and have not only built their own infrastructure on top of it but created a line of business in Amazon Web Services to sell a set of services, all of which run on open source software on some level, to other companies. It is, however, very reluctant to disclose details in terms of its usage, and is not a major contributor to open source more broadly. From this, it is easy to conclude that the company believes that software innovation is still worth protecting. This semi-opaque model differentiates companies of this generation from the fourth, or current, generation of software creators.

Fourth Generation (2004–present)

Like Google, the group of Facebook, GitHub, LinkedIn, and Twitter are all principally built on open source software as opposed to proprietary alternatives, in large part because the economics of licensing software at extreme scale re-

main problematic. Unlike Google, however, Facebook, GitHub, LinkedIn, and Twitter tend to operate as if internally developed software is less of a differentiator or competitive advantage. Each has released sizable internally created projects as open source. GitHub founder Tom Preston-Werner summed up that company's justifications in a 2011 piece, "Open Source (Almost) Everything," detailing the perceived benefits of open sourcing noncore assets, which include better efficiency in hiring and retention, improvements in visibility, less duplication of effort, and more. What these and other justifications imply, however, is simple: in cases where source code does not represent a competitive advantage—which is most cases in most companies—the benefits of releasing source code far outweigh the costs of keeping it proprietary.

We have come full circle, in other words. Software, once an enabler rather than a product, is headed back in that direction. There are and will continue to be large software licensing revenue streams available, but traditional high margin, paid upfront pricing will become less dominant by the year, gradually giving way to alternative models we'll discuss later in this book.

How Did This Happen?

Introduction

The most obvious question facing an industry built on core assumptions of intrinsic software value is: how did this happen? How did an asset that has generated trillions in profits become gradually devalued? The answer to that question is complicated, but the evidence has been there for some time.

The Challenge of Competing with Free

One of the most obvious factors acting as a brake on software pricing has been the wider availability of open source software. As the *Encyclopedia Britannica* discovered with Wikipedia, it is difficult to compete with free, even in cases where the premium product is technically superior or differentiated in a meaningful way. Likewise, proprietary software vendors have been forced to adapt to a library of open source alternatives growing by the hour.

Organizations seeking to commercialize open source software realized this, of course, and deliberately incorporated it as part of their market approach. In a 2013 piece on Pando Daily, venture capitalist Danny Rimer quotes then-MySQL CEO Mårten Mickos as saying, "The relational database market is a $9 billion a year market. I want to shrink it to $3 billion and take a third of the market." While MySQL may not have succeeded in shrinking the market to three billion, it is interesting to note that growing usage of MySQL was concurrent with a declining ability of Oracle to sell new licenses. Which may explain both why Sun valued MySQL at one third of a $3 billion dollar market and why Oracle later acquired Sun and MySQL. The downward price pressure imposed by open source alternatives have become sufficiently visible, in fact, as to begin raising alarm bells among financial analysts.

> *The legacy providers of data management systems have all fallen on hard times over the last year or two, and while many are quick to dismiss legacy vendor revenue shortfalls to macroeconomic issues, we argue that these macroeconomic issues are actually accelerating a technology transition from legacy products to alternative data management systems like Hadoop and NoSQL that typically sell for dimes on the dollar.*
>
> *We believe these macro issues are real, and rather than just causing delays in big deals for the legacy vendors, enterprises are struggling to control costs and are increasingly looking at lower cost solutions as alternatives to traditional products.*
>
> **— PETER GOLDMACHER**
> Cowen and Company

Even hardware companies are at risk. Cisco has built itself into a large systems supplier primarily on the back of its networking business; switching and routing are typically responsible for close to two thirds of the company's revenue. For years, even as areas such as compute and to a lesser extent storage began to succumb to the onslaught of software-powered alternatives such as the public cloud, networking remained relatively immune. With the rise of software-defined networking on the horizon, however, networking giants will be forced to compete with a new breed of player, one that like Mickos before it wants to attack a large industry, shrink it, and siphon off a portion of the profits. What would this mean for Cisco specifically? According to one report, CEO John Chambers asked a few of his senior executives just this question. Their answer? A move to SDN would turn Cisco's "$43 billion business into a $22 billion business."

Nor is any respite in sight. What began as a trickle of open source software with the early success of the Linux, Apache, MySQL, and PHP (LAMP) stack has lately turned into a flood. The number of software categories without a viable open source choice is growing smaller by the day. As Donald Knuth recently put it in an interview:

> *The success of open source code is perhaps the only thing in the computer field that hasn't surprised me during the past several decades. But it still hasn't reached its full potential; I believe that open-source programs will begin to be completely dominant as the economy moves more and more from products towards services, and as more and more volunteers arise to improve the code.*
>
> **— DONALD KNUTH**
> Professor Emeritus at Stanford University

From modern technology organizations committing to open sourcing the majority of their nonstrategic codebase to companies using open source as an asymmetrical means of market competition to software developers making their solutions to various problems available to the wider world, the rapid growth of open source has forced commercial software vendors to adapt both their products and the models with which they sell them. In many cases, open source software has also impacted pricing.

So if open source is cannibalizing the commercial software markets, it's all smooth sailing for those who commercialize open source, right? Well, not exactly. In order to monetize an otherwise free and open source software product, vendors have been forced to develop creative new business models to get buyers to pay for what they can otherwise get for free. The most common of these are described below.

Support/service

The most common model of commercial open source is support and service. Instead of paying for the product, buyers pay vendors to support a product they can otherwise obtain at no cost. The advantage of this model is that most large organizations require commercial support for production applications, so sales is less of a challenge. The disadvantage of this services-only approach is that the deal size is commensurately lower than with traditional commercial software that includes both a license component and support and service.

Dual licensing

Another popular model historically has been dual licensing. Best exemplified by MySQL, this model requires that a relatively restrictive open source license, typically the GPL, be applied to a given codebase. Unlike so-called permissive licenses such as Apache, BSD, or MIT, the GPL and other similar "copyleft" licenses require that any changes, modifications, or additions to a given codebase be made available under exactly the same terms. For commercial organi-

zations that wish to embed open source in a closed source product, this could mean being forced to open source code that would prefer to remain proprietary. Rather than comply, then, buyers can purchase an alternate, hence the "dual," license for the product, which allows them to keep their code private without requiring anything in turn. While this model spawned a host of imitators, however, usage of it has been declining for some time. For the model to work, the vendor must maintain or acquire copyright permissions for the entirety of the codebase, and as MySQL itself has proven, this can prove problematic over time as would-be contributors make fixes and updates available but decline to share copyright with the vendor.

Open core/hybrid source

Probably the most common model today, open core describes an open source project that is partially, even mostly, open source, but with some portion of the project or some features remaining proprietary. Typically there is a basic level of functionality—referred to as the core—which remains open, and proprietary features or capabilities are added upon and around this. The highest profile example of this model today is Hadoop. Cloudera, the first organization to commercialize the data processing platform, contributes along with other organizations, commercial and otherwise, to the base Hadoop project, which is open source. A proprietary product that includes management functionality is then sold to customers on top of the base open project. This model is viable, but can be difficult to sustain. One of the challenges for those adhering to the open core model is that the functionality of the underlying open source project is evolving at all times, which means that the proprietary extensions or features must outpace the development of the open source project to remain attractive to customers.

The reality then for purveyors of commercial open source offerings is that while the models have been demonstrated to work, in some cases generating substantial market value, none work particularly well. Each model has limitations that act to inhibit the types of growth we have seen from the software market in years past. It's commonly accepted, in fact, that the market will never see another Red Hat, which is to say another billion-dollar revenue entity that primarily or exclusively sells open source software. And even those responsible for open source businesses admit that open source is problematic from a business model standpoint. As Cloudera founder Mike Olson articulated in an essay for LinkedIn:

> *The moral of that story is that it's pretty hard to build a successful, standalone open source company. Notably, no support- or services-only business model has ever made the cut. Red Hat, the apparent exception, isn't: The company rode its closed-source Red Hat Network offering to dominance, effectively crushing the competition before releasing that hosted infrastructure as open source in 2008....*
>
> *So here is the conundrum facing enterprise infrastructure software companies:* ***You can no longer win with a closed-source platform****, and* ***you can't build a successful standalone company purely on open source****. [emphasis his]*
>
> **— MIKE OLSON**

If open source is challenging proprietary software companies, and also those wishing to commercialize the open source software, the real moral of the story might be that it's pretty hard to build a successful, standalone software company, period.

The Challenge of Competing with Available

In addition to the challenge of competing with software that is freely available, commercial software vendors are increasingly pitted against software that is freely consumed as a service. Certainly this was the view of then-Microsoft CTO Ray Ozzie, who as far back as 2005, had told the software-focused company that "the most important step is for each of us to internalize the transformative and disruptive potential of services." Unfortunately for Ozzie, it wasn't until his departure in 2010 that Microsoft appeared to be fully putting its weight behind that message with the launch of Azure.

In the early years of the last decade, vendors such as Salesforce were building traditional business applications that were not designed to be shipped to and installed at a customer's premises, but rather hosted remotely and accessed via a browser. While the browser technology of the time had its limitations, the model's convenience more than offset any functional issues with the platform. Over time, the browser-based delivery model became not only accepted, but the default. Even software that is installed and hosted on-premise today is more often than not consumed via a browser; native applications are increasingly rare outside of mobile.

More recently, other categories of software such as databases have been made available as services. Amazon and Google, for example, both host versions of the open source MySQL database that users may leverage via an API. Both companies

also offer more specialized database services in Redshift and BigQuery, respectively. Beyond database-style services, Amazon and Google, along with a wide market of other public infrastructure providers, offer a range of compute, storage, and other software-based services.

What this means for commercial software providers then is that would-be customers are increasingly able to choose between two options: software they are responsible for selecting, purchasing, installing, customizing, integrating, deploying, and maintaining, or software they simply consume as a service. While not yet appropriate in every customer scenario, the number of problematic use cases for SaaS is growing smaller by the day. Which in turn means that the competitive pressure on standalone, on-premise software is increased, because it is inherently more difficult to consume than SaaS alternatives.

The Challenge of Competing with Your Customer

Two decades ago, businesses needing a way to persist data were most likely to turn to one of the major relational database suppliers: IBM, Microsoft, or Oracle. But as users' demands for innovation grew, vendors were unable to keep pace. As Adam Bosworth, a former employee of Microsoft, Google, and BEA wrote in 2004:

> *About five years ago, I started to notice an odd thing. The products that the database vendors were building had less and less to do with what the customers wanted. This is not just an artifact of talking to enterprise customers while at BEA. Google itself (and I'd bet a lot Yahoo!, too) have similar needs to the ones Federal Express or Morgan Stanley or Ford or others described quite eloquently to me.*
>
> **— ADAM BOSWORTH**

Whether driven by an inability of vendors to meet their needs, the high cost of the proprietary software, or both, the end result was that users with the means to create their own databases did so. From Google's Dremel, Pregel, and Spanner to Facebook's Cassandra, Hive, and Presto, an entirely new ecosystem of software was created by businesses that do not sell software. Nor was this roll-your-own trend limited to databases; Git, the distributed version control system behind GitHub, was written by Linus Torvalds to replace a commercial alternative. Rails, the popular Ruby-based software framework available on PaaS platforms like Heroku, was originally extracted from an existing SaaS product from 37Signals called Basecamp. Even companies like GE are helping to fund noncommercial software, having contributed $105 million to Pivotal, the home of projects like Cloud Foundry. The first

version of the OpenStack project, meanwhile, was created using code from both NASA and Rackspace.

Not every business has the resources or capability to build software that would compare favorably to commercial alternatives, of course. But fortunately for less technically capable entities, a great deal of the software written to solve problems within an organization is subsequently released as open source software. With the exceptions of Dremel, Pregel, and Spanner—about which there are papers documenting the technology and approach—every example mentioned was released as open source software. When users are able to help other users solve technology problems with software, the attendant commercial opportunity for vendors becomes more problematic. It means either competing with free, which as discussed is enormously difficult, or attempting to monetize free by commercializing open source software, which is possible but growing more difficult.

All of which helps explain why, as we'll see later, many commercial organizations are aggressively diversifying their revenue streams by expanding from software into services.

The Challenge of Developer Empowerment

The biggest potential challenge for commercial software organizations, however, is the developer. As the availability of source code and services has democratized access to technology and irrevocably altered traditional procurement processes, developers have increasingly become technology kingmakers. Where technology acquisition was once the province of the CIO, today it's the practitioner leading that process, because by the time a CIO typically hears about a project today, a majority of the technology and architectural decisions have already been made.

To be sure, this is not the first time we have seen a major shift within organizational technology buyers. As technology grew more common within enterprises, for example, responsibility for purchasing and procurement gradually moved from traditional operational elements to departments that specialized in technology usage and deployment—what we today know as IT. In more recent years, meanwhile, IT's control of technology usage and adoption has been waning, frequently in favor of more empowered marketing departments.

But from the standpoint of a commercial software vendor, there is one key difference between developers and other organizational bodies that have controlled technology acquisition in years past: developers typically have no budget. Which means that the single most important audience from a technology adoption stand-

point frequently has limited or no ability to pay for the products a commercial software vendor is offering.

Most commercial software vendors, though slow to wake up to this new reality, are beginning to move aggressively to court developer populations and update their engagement and outreach capabilities. Instead of relying strictly on an enterprise-focused sales force, armies of technical evangelists and developer engagement professionals are being unleashed on unwitting developer populations in an attempt to ensure a given software vendor's relevance for the population most likely to be making technical decisions.

The problem facing software vendors, however, is that while recognizing the problem is indeed the first step, the solution is less than obvious. Commercial software vendors are particularly disadvantaged, because they are compelled to compete in a two-front war with both free software and software made available as a service. From a competitive standpoint, this means downward pressure on price with potentially higher costs driven by a need to compete with different models.

But as discussed, even commercial open source vendors are challenged by a market that is increasingly valuing convenience and availability over performance and features. Developers have long prized availability and speed, and while open source software is preferred, open source software that is managed by a third party possesses crucial advantages in terms of convenience.

The Software Paradox at Work

Because the implications of the Software Paradox are wide-ranging, it is important to study organizations that differ in both size and context to get a broad understanding of the potential impacts. In this chapter, we'll look at how very different companies are looking at the value of software, both in the commercial and functional senses of the word. Some will serve as cautionary tales, others will provide constructive feedback for how to view software moving forward. All, however, should be useful for testing your own assumptions about what software is worth.

Adobe

In the year 2000, it was difficult to foresee just how powerful the idea of delivering Software-as-a-Service—a model in which browser-based applications replaced their native, operating system–specific counterparts—would become. Even following the initial public offering of Salesforce in June of 2004, skepticism remained regarding the existential threat the browser posed to native application development. Eventually, however, it became clear that this was not only a viable model, it was an inevitable one for most applications. Today, whether an application's backend is hosted on premises or remotely, the overwhelming majority of interfaces are consumed via a browser.

There is one notable exception to this trend, however: Photoshop. It was such an obvious counterpoint that "everything short of Photoshop" became a cliché among industry watchers discussing the ascent of the browser-based software delivery model. Photoshop, and the other portions of what Adobe eventually packaged and marketed as the Creative Suite, was both sold and delivered as a traditional standalone software application. Buyers purchased a perpetual license, downloaded and installed a closed source software product, and the model changed little in spite of the tectonic shifts in the software world around it.

Today, this is largely still true. Photoshop did have a limited-functionality edition become available in the browser, but the majority of its users still consume the native versions of the product. Adobe did make one important change in 2013, however. Instead of selling perpetual licenses to a shipped product, it transitioned the majority of its user base to monthly subscriptions. This may seem like a difference without a distinction, but the implications for Adobe and its users were profound.

Instead of buying Photoshop, for example, for $650 outright, Adobe sold subscriptions to the software for $20 a month. At this subscription level, Photoshop users would have paid the full retail price over a period of roughly 32 months. Presumably to upsell buyers, the payback period of the full suite—normally priced at around $2,600 but sold in a subscription format at $50—was 52 months rather than the less than three years for Photoshop.

The most common reaction to this sea change in Adobe's business model was outrage. Design professionals penned scathing reviews on their blogs, and comment sections on media items covering the change were one vitriolic comment after another. One user even went so far as to create a petition on Whitehouse.gov asking the Obama administration to look into the subscription pricing, alleging that it was predatory in nature. (The petition did not accumulate enough signatures to meet the response threshold and was removed.)

While a great deal of the anger cited cost as the primary complaint, then-columnist for the *New York Times* David Pogue wrote in *Scientific American*:

> *Paying a monthly fee for software doesn't feel the same. We download a program, and there it sits. Month after month we pay to use it, but we get nothing additional in return.*
>
> **— DAVID POGUE**

This assertion is debatable, because aside from evening out Adobe's cash flow, a monthly subscription model allows vendors to develop iteratively rather than withholding these features to incent the purchase of major upgrades every few years. It's easy to make the argument, in fact, that companies like Adobe shifting to this model must develop this way to limit churn.

But even assuming no changes in Adobe's development model, a subscription model is by definition more accessible for casual users, and like public cloud pricing, may even allow capital-constrained designers to amortize the product's cost over a longer period of time. From the level and tone of public rhetoric, however,

it would appear that these proposed advantages were less than apparent to the majority of users.

The psychology of the price change would predict, in any event, a major downturn in Adobe's business, accompanied by perceptible shifts toward even more functionally limited or harder to use but free alternatives such as the Gimp. The crucial question for Adobe—and though they may not have been aware of it, other vendors of shrink-wrapped software—was simple: did this happen?

The short answer seems to be no. As summarized by *Bloomberg*'s Joshua Bruste in March of 2014:

> *Adobe now has 1.8 million customers paying for these software subscriptions, and it added 405,000 in the last quarter, the company said on Tuesday in its quarterly earnings report. It is making more money selling monthly subscriptions to its Creative Cloud software —the family of programs that includes Photoshop and Illustrator— than it is by selling the software outright.*
>
> **— JOSHUA BRUSTE**

By making the change to a subscription model, Adobe effectively abandoned its traditional model—the same traditional model of software sales that built it into a $30 billion company—in favor of recurring revenue. This isn't a change made lightly, nor one without costs: by transitioning to subscription revenue, Adobe is deliberately extending its revenue recognition period out over time while taking an upfront capital expense hit. Which, in turn, tells us several things. First, that Adobe expects the profits on a customer's lifetime value to exceed the return it could realize from the revenue up front. Second, that such a model has the potential to expand Adobe's addressable market by creating an avenue for casual users to become paying customers, if only for a brief period. And third, and perhaps most important, the shift in the model is a signal that the days of shrink-wrapped, paid upfront software are nearing an end.

Amazon

Incorporated in 1994 as Cadabra, Amazon.com debuted to the world in 1995. At the time, Amazon was focused strictly on the online sale of books. Over time, as Amazon outlasted a sea of competitors online and offline, the merchant's scope expanded dramatically to what we know today: Amazon as a retailer of virtually every type of good, physical, virtual, and otherwise. It was this identity, Amazon as a mere retailer, that persisted for years even after the company transparently en-

tered the technology services market and began competing directly with the industry's massive incumbents.

When Amazon launched its Elastic Compute Cloud (EC2) and Simple Storage Service (S3) in 2006, most of its competitors at the time considered them "toy" applications, academically interesting but of no real import. Most focused on what the services could not do, and in their initial incarnations, that list was long. Comparatively fewer observers paid attention to what Amazon could do: spin up compute and storage instances in 90 seconds or less, and charge for them by the hour to anyone holding a credit card. But who was going to make money charging pennies on the hour? As one senior technology executive responded when questioned about Amazon at the time, "I don't want to be in the hosting business." Even technology companies directly in Amazon's path seemed unable to perceive the threat and react accordingly. As Microsoft's Ray Ozzie acknowledged in an interview in 2008, "[the cloud market] really isn't being taken seriously right now by anybody except Amazon."

Eight years later, and Amazon is now correctly regarded as the dominant player in one of the most important emerging markets in the history of the technology industry. Like Microsoft in operating systems or office productivity software or VMware in virtualization, Amazon is the vendor whom other players must relate themselves to in some way; whether it's as a sanctioned API-compatible ally in Eucalyptus or an open source alternative in OpenStack. Unlike Microsoft and VMware, however, Amazon sells no software. Or more accurately, it sells no software in the traditional distributed fashion. Besides its existing software-powered infrastructure businesses in compute, storage, and so on, Amazon also sells existing software products as a service. From discrete databases like MySQL, PostgreSQL, and SQL Server to packaged, all-in-one offerings such as Amazon Redshift (which the company recently said was the fastest growing service in the history of AWS), Amazon deliberately eschews labels like IaaS or PaaS but is the pre-eminent service business.

As such, the company's value from a technology perspective isn't software, strictly speaking, but rather outsourced effort. Any business can download and run software like MySQL or PostgreSQL at no cost. But hosting it, keeping it up and running, backing up the databases, and exposing them safely to other applications requires expertise and effort. For many customers, and AWS customers in particular, then, the value isn't in the software itself—because that is available at no cost—but the saved expertise and effort of consuming the infrastructure software as a service.

Amazon, in other words, is making money with software, rather than from software. This may seem like a difference without a distinction to some, but it is actually an excellent illustration of one path forward for software monetization, and thus for mitigating the Software Paradox. By combining software with another, more readily monetized product—services, in this case—Amazon is able to efficiently extract profit from a growing, volume market. What's more impressive, however, is that because Amazon is building primarily from either free software (in the economic sense) or software it developed internally, it is paying out minimal premiums to third parties for the services it offers. Which means that not only is AWS a volume business, it may be a high-margin business at the same time. Amazon does not break out its AWS revenues, so we're forced to rely on estimates, but UBS analysts Brian Fitzgerald and Brian Pitz projected in 2010 that AWS's margins would grow from 47% in 2006 to 53% in 2014. Last year, Andreas Gauger, the chief marketing officer for Amazon competitor ProfitBricks, estimated Amazon's margins were better than 80%.

If Gauger is correct, that would put AWS in software territory from a margin perspective; IBM's software margins in 2013, for example, were 88.8%. But even if the more conservative UBS figures are closer to the mark, AWS has found a reasonable middle ground between volume and margin. And if one assumes the Software Paradox to be correct, this is particularly true, because the high end of software margins is likely to be unsustainable. By leveraging software to generate revenue without having to sell it directly, however, AWS is inherently hedged against this prospect.

Apple

While it's easy to forget today, given the company's unprecedented success and growth in recent years, for many years, Apple's approach was considered a case study of how not to operate a technology business. For the better part of its early existence, Apple had resisted calls to license its software to other manufacturers, even as Microsoft grew explosively on the backs of just that model. In describing why he left Apple for Microsoft in 1981, Jeff Raikes told a group of Albers School of Business students in 2004 that "I wasn't sure who was going to win in the hardware business, but it sure looked like Microsoft was doing the software for all of them."

The implication was simple: when software is where the value lies, it's incumbent on businesses to maximize the addressable market for that software by making it available on whatever hardware it's practical to run it on. Apple, and more specif-

ically Steve Jobs, believed that it was the combination of Apple hardware and software that delivered the kind of experience users expected. The market, however, did not share that opinion, propelling Microsoft to lofty heights and punishing Apple at the same time.

Predictably, this led the post-Jobs Apple to experiment with licensing its operating system to other manufacturers in an attempt to increase penetration. For two years in the mid-1990's, manufacturers from Bandai to Motorola were producing Mac clones with Apple's blessing. This decision was effectively killed by Jobs following his return in 1997, and to this day, Apple has never returned to a software licensing model, with rare exceptions such as CarPlay.

As during his original tenure, Jobs pursued an integrated software and hardware model as opposed to software alone. The results some 16 years after Jobs returned to the company he founded will not surprise current followers of the company. In June of 2010, Apple's market capitalization surpassed that of its longtime nemesis Microsoft, and the one-time underdog has become instead the biggest player in an industry full of them.

Why did the Jobs strategy fail initially only to succeed years later? Some of it undoubtedly is attributable to execution on both companies; Microsoft was astute in parlaying its overwhelming success in one market (operating systems) into dominance in another (office productivity). Apple, meanwhile, stalled after investing without focus in markets from digital cameras to tablets (though they would obviously have the last laugh there). But the most important factor in the shift may have less to do with either company and more to do with the wider market context.

In the nearly two decades that Microsoft dominated Apple, the underlying software platform was an all-important consideration because of application lock-in. It was difficult for Apple's operating system to achieve mainstream traction because it was perpetually caught in a chicken-or-egg situation with respect to applications. There weren't enough applications for mainstream consumers to adopt the platform, and because there weren't enough mainstream consumers, the incentive for application manufacturers to support the Mac along with Windows was low. Two important shifts altered the fundamentals of this market enabling Apple's ascent.

While the trend had begun considerably beforehand, the launch of Gmail and the initial public offering of Salesforce in 2004 together heralded the arrival of browser-based applications. It would take years for the default desktop application to become browser based rather than native, but the writing was on the wall. For would-be adopters of Apple's hardware, the rise of browser-based applications was

transformative. Because browsers supported the major available operating systems, it no longer mattered as much what the underlying operating system was. Serve your application up via a browser and it didn't matter whether a user was on Windows, Mac, or even Linux. Coupled with Apple's aesthetically superior user interface, this gave Apple new life in its battle against Windows hegemony.

Even still, the sheer inertia behind Windows would be difficult to overcome. So Apple instead applied innovative software and hardware to net new markets—MP3 players first, then smartphones, and finally tablets. In none of these markets did Apple license the software to exterior manufacturers; the value instead accrued to the company that could combine unparalleled software design expertise with a growing competence in supply chain management (courtesy their current CEO, Tim Cook). Interestingly, Apple's strategy relied initially on the browser-based application delivery model before replicating, effectively, the native lock-in strategy Microsoft had previously leveraged to great effect against Apple. When the iPhone was launched in 2006, it launched without any ability for companies other than Apple to add applications to the platform. A year later—against Steve Jobs' initial preferences, according to some reports—Apple announced the availability of its SDK. Billions of applications later, Apple now benefits from the same application investment protection that Microsoft once enjoyed with Windows.

The lesson here is simple: software is an important piece of the equation, but only a piece. It's difficult to imagine Apple, for example, enjoying the same financial success that Microsoft enjoyed with Windows had they licensed the platform to third parties. Google's Android project is the closest to replicating the Windows model in the mobile world, and indeed is now the volume leader over iOS in most markets, but it has never tried to monetize the platform directly, and any such attempt would have likely prevented its current market dominance. Android is similarly valuable to Windows strategically, it's simply worth billions less in licensing fees.

Whether we're looking at Apple or Google, then, the lesson is that software is once again a means to an end as opposed to an end in and of itself. Which is a profound statement about the commercial value of the software powering the devices that we all use today.

Atlassian

Founded in 2002 by Mike Cannon-Brookes and Scott Farquhar, Atlassian was a small Australian software company bootstrapped on $10,000 in credit card funding. By 2011, the private company announced its annual revenues had eclipsed $100

million. As of March 2014, the company had 35,000 organizations using its products and was valued by the private market at $3.3 billion.

This growth trajectory is impressive but hardly unprecedented by technology industry standards; at the same age, for example, Microsoft was worth over $20 billion, not adjusted for inflation. Atlassian has achieved this valuation, however, in spite of some potential strategic limitations.

Most famously, Atlassian doesn't employ sales people. This likely isn't technically true, as they report 15–20% of annual revenue is applied to sales and marketing resources. But there is no question that relative to its peers, Atlassian is substantially under-resourced with respect to salespeople. Instead, Atlassian has relied on clear, accessible pricing, bottom-up adoption, and developer-enabled word-of-mouth to progress from a two-person startup to an 800-strong, pre-IPO software vendor.

More interesting, however, was the nature of the product itself. When Atlassian was founded in 2002, selling strictly proprietary software was not unusual. Quite the contrary: Red Hat, one of the few commercial open source vendors at the time, had only been a publicly traded entity for three years. By 2014, however, it became more and more unusual for organizations not to make some portion of their portfolio open source for competitive, strategic, or simply practical reasons. Atlassian, however, has not chosen to leverage open source as part of their strategy, possibly because they haven't had to. Sales growth has clearly not been an issue, and the company has generated goodwill in a variety of communities due to their decision to make their software available for free for open source projects.

Growth aside, however, it's difficult to imagine the company achieving its $3+ billion valuation without the October 2011 addition of OnDemand. Prior to its introduction, the majority of the Atlassian product catalog was made available as software-only, rather than as a service. This meant that the burden of installation, configuration, and maintenance was principally on the user. With OnDemand, Atlassian made virtually its entire product catalog available as a remotely hosted and managed service.

It has become increasingly important for all companies to be capable of delivering their software as a service, to cater to customers that prefer to consume it in that fashion. But this was even more true of Atlassian. Given the base for its typical adoption cycle, in which one or more components are acquired by an individual or small team, minimizing the friction of adoption is enormously important. SaaS is an ideal way to accomplish this, because it shifts the operational burden from user to vendor.

What this means for Atlassian moving forward is that if the observed downward trajectory of available commercial software licensing revenue in the wider market continues, the company is inherently hedged with its OnDemand line. Longer term, as well, the company stands to gain by having operational visibility into a growing percentage of its user base's implementations. Instead of having its customers' usage patterns of the software be opaque, Atlassian can collect and store telemetry on hundreds of thousands of running instances to inform its product planning and support, and potentially even retailing the aggregated and anonymized data back to customers—thus creating a new revenue stream from what is essentially waste data today.

IBM

While it's clear in retrospect that IBM misjudged the potential of the software market and left an enormous amount of value on the table for Microsoft, the technology giant did adjust its strategy over time. IBM Software Group (SWG) is the business unit principally responsible for its standalone software—and as of 2010, solutions—product lines, and while software accounted for only 26% of its 2013 revenue, it contributed 47% of its total profit. IBM's long-term goal, in fact, is for the majority of its profit to be derived from its software business.

To get an understanding of how realistic that is, it's worth looking at the software group's revenue growth over the last five years. The Compound Annual Growth Rate (CAGR) for the period is just shy of 5% (4.92%), but if we look at the data by year, we can better evaluate actual trajectory, as shown in Table 4-1.

Table 4-1. IBM SWG revenue and profit growth: 2009–2013

YEAR	REVENUE GROWTH	PROFIT GROWTH
2009	-3.1%	.6%
2010	4.84%	1.37%
2011	9.86%	0.68%
2012	1.98%	0.23%
2013	1.87%	0.11%

The decline in 2009 is best viewed as an anomaly, given the context of the global financial crisis that negatively impacted spending across the technology industry. This decision is validated by the fact that, as with many of its technology industry counterparts, IBM rebounded nicely for two consecutive years following the panic. Since that time, however, it's become apparent that software revenue

growth has stalled in recent years. Certainly growth is preferable to declines such as 2009, but for a business division expected to be a major engine of growth for the company, it's less impressive.

There are many potential explanations for the less-than-robust performance, but IBM's current strategy suggests that one component at least is a challenge to the traditional shrink-wrapped software business.

As much as any software provider in the industry, IBM's software business was optimized and built for a traditional enterprise procurement model. This typically involves lengthy evaluations of software, commonly referred to as "bake-offs," followed by the delivery of a software asset, which is then installed and integrated by some combination of buyer employees, IBM services staff, or third-party consultants.

This model, as discussed previously, has increasingly come under assault from open source software, software offered as a pure service or hosted and managed on public cloud infrastructure, or some combination of the two. Following the multi-billion dollar purchase of Softlayer, acquired to beef up IBM's cloud portfolio, IBM continued to invest heavily in two major cloud-related software projects: OpenStack and Cloud Foundry. The latter, which is what is commonly referred to as a Platform-as-a-Service (PaaS) offering, may give us both an idea of how IBM's software group is responding to disruption within the traditional software sales cycle and their level of commitment to it.

Specifically, IBM's implementation of Cloud Foundry, a product called Bluemix, makes a growing portion of IBM's software portfolio available as a consumable service. Rather than negotiate and purchase software on a standalone basis, then, IBM customers are increasingly able to consume the products in a hosted fashion. And in case it wasn't clear how seriously IBM is committed to this directionally, the company publicly committed to investing a billion dollars in the Cloud Foundry ecosystem. This is not a first for IBM; the company has committed a billion to projects before, most notably with Linux. But as a signal of intent, it is notable that the company is making a Linux-sized commitment to an application layer that is optimized for making applications portable between many types of environments, including on-premise to cloud.

Obviously some, perhaps even a majority, of the company's customers will prefer to acquire and implement software in the traditional manner. But if IBM's revenue fortunes are any guide, its network-enabled strategy can't be rolled out quickly enough.

Nest

Five years ago, people who claimed that a company producing two products—and unloved products at that, in thermostats and smoke detectors—would exit for $3.2 billion would have been laughed at and publicly mocked. Today, they might instead have been regarded as witches, because that's exactly what Google paid in the transaction that landed them Nest. How did the company accomplish this?

Most credit, at least in part, the design acumen of Tony Fadell, who is commonly regarded as one of the fathers of the original iPod, which indeed the Nest bears more than a passing resemblance to. Fadell, this argument goes, built a team that was able to emphasize aesthetics and usability, which when combined with intelligent software, produced a product unlike anything the market had seen before. It was an industrial device with legitimate aesthetic appeal.

The most important decision Nest ever made, however, might have been to become what Chris Dixon, an investor with Marc Andreessen's Andreessen Horowitz venture firm, describes as a "full stack startup." Nest could certainly have pursued a less ambitious and lower risk strategy of developing its learning software and user interface, then license it to organizations with greater logistical and manufacturing capabilities but less competence within software.

The problem with this approach is that, by sacrificing control of the stack, you introduce opportunities for partners and customers to negatively impact the overall offering. Consider the difference between a Curb, for example, and an Uber: the latter controls the experience top to bottom, and is thus generally able to deliver a superior experience. Dixon lists multiple areas of concern for startups pursuing less than a full stack approach:

- *Bad product experience. Nest is great because of deep, Apple-like integration between software, hardware, design, services, etc., something they couldn't have achieved licensing to Honeywell, etc.*
- *Cultural resistance to new technologies. The media industry is notoriously slow to adopt new technologies, so Buzzfeed and Netflix are (mostly) bypassing them.*
- *Unfavorable economics. Your slice of the stack might be quite valuable but without control of the end customer it's very hard to get paid accordingly.*

— CHRIS DIXON

This argument, then, is establishing a value for software that is dependent on context. As a standalone offering, the software that makes the Nest a class-leading device has value. But it is likely a fraction of the value that can be realized by combining that software with other components—hardware and backend services, in Nest's case—to create an interesting and distinct whole. *E pluribus unum*, in other words.

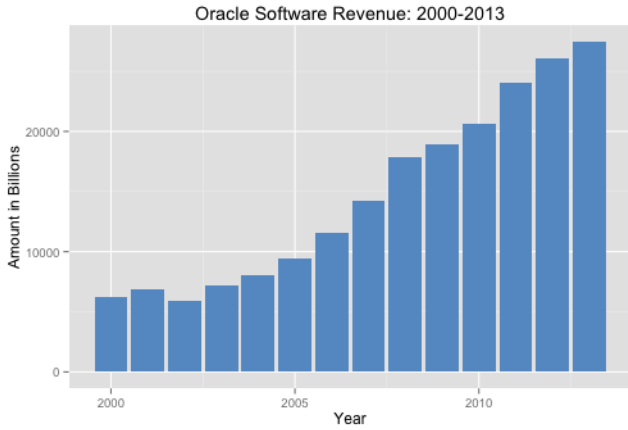
This value is challenging to realize, of course. Achieving operational excellence in categories above and around software is exponentially more difficult to achieve than in software alone. But the rewards are much greater, and in many cases, the Software Paradox may mean that would-be providers of software have no choice but to expand their product focus. Nor has this trend gone unobserved by investors; per the *Wall Street Journal*, "US venture capitalists completed a record 31 fundraising deals for consumer-electronics makers last year, eclipsing the previous high of 29 in 1999, according to DJX VentureSource. They pumped $848 million into hardware startups, nearly twice the prior record of $442 million set in 2012."

Although the greater risk and challenge of a full stack startup may be necessary, it also offers the potential for much higher upside. In the margins on the actual product, yes, but also in under-realized and under-appreciated ancillary lines of business. In the case of Nest, this is data. As we'll see later, this is by itself an enormous potential revenue engine.

Which is why it's logical to expect this trend to continue as VCs and startups alike grapple with a shrinking outlook for software-alone startups.

Oracle

On paper, the story of Oracle's software business is a success story. Not only has software, which accounted for 74% of the company's revenue in 2013, built Oracle into one of the largest businesses in the world, it has demonstrated a consistent ability to grow revenue even amid challenging economic conditions. In every year of the last decade, Oracle has bettered its software revenue of the year prior. Revenue growth was more limited in some years than others, such as during the global financial crisis, but still present in spite of such catastrophes. The following chart depicts that growth.

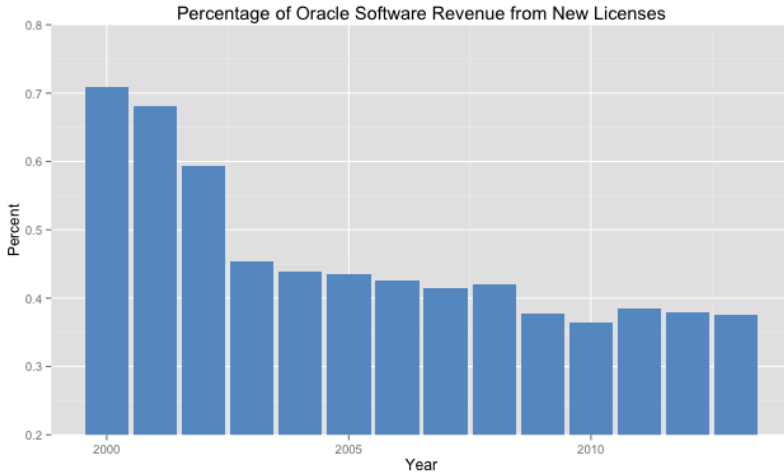


This, for Oracle, is the good news. The bad news is that if one looks beneath the surface, there are questions about how sustainable this growth is over the longer term.

In a fashion relatively unique among large technology organizations, Oracle provides an unusual level of detail in its financial information. Among other distinctions, Oracle makes available to the public the precise breakdown in revenue between what it terms "new software licenses and cloud software subscriptions" (note: the inclusion of "cloud software subscriptions" is new as of 2013) and "software license updates and product support." What this means in practical terms is that we can attempt to discern how much of Oracle's revenue derives from new customers versus its monetization of existing customers, and what this trajectory looks like over time.

In its 2013 financial year, for example, Oracle's filings state that $10.3 billion of its $27.5 billion in total software revenue came from the sale of new licenses and, in a new development, "cloud software subscriptions." But how does this number

compare to Oracle's past history in the sales of new licenses? Unfortunately for the company, the answer is: not well.



In the year 2000, about 71 cents of every dollar Oracle generated from software revenue came from the sale of a new license. As of 2013, that figure was down to less than 38 cents. Which means that in less than 15 years, Oracle's software revenue has shifted from almost three quarters from new licensing to just over a third. One logical explanation for this is that basic arithmetic says that growth is hard to scale. While it's simple for a startup to double its revenues, for example, it is not realistic to expect a company of Oracle's size to accomplish the same feat. But attributing Oracle's decreasing ability to sell new perpetual licenses strictly to the revenue plateaus large companies inevitably face as their market becomes saturated would be a mistake. In the year 2000, when it was still posting new licensing figures above 70%, Oracle had already been selling databases for well over two decades.

Oracle's decision in 2005, after two years of major drops in new license sales, to sell all-you-can-eat Enterprise Licensing Agreements clearly and demonstrably accelerated its revenue growth. But this somewhat artificial growth came with a catch; the audit teams that are a product of the ELAs are a constant reminder for customers to consider other, less costly (and unmonitored) alternatives, further depressing the sales of new licenses. Which means that growth must come from an ever-shrinking pool as customers driven by costs, functionality, or both aggressively examine other options. Even for a company able to consistently and impressively generate revenue growth from the sale of software, this is a problematic trend.

While the causal mechanisms are more difficult to prove, it's certainly plausible that this trend in the area of its business that makes the largest contributions to its revenue pool is contributing to Oracle's repeated misses of financial estimates over the last two years. Even if it's not directly responsible, it certainly is not helping to mitigate the difficulties Oracle is having competing in a rapidly changing landscape. At the very least, it's a more plausible explanation than Oracle's famously aggressive sales force having suddenly been afflicted with what CFO Safra Catz characterized as a "lack of urgency" in March of 2013.

The irony of the Software Paradox for Oracle is that on paper, the company should be booming. As hundreds of new applications come online and struggle to scale to meet the demand of millions of new users, the opportunities for a battle-tested, production-quality database should be virtually limitless. But in a market where commercial software is worth less than it once was and customers have lower cost and more available options, you'd expect to see stalling license sales and a decreased ability to meet analyst expectations. What we're seeing at Oracle, in other words.

To its credit, the company is responding to some of the more notable disruptive challenges it's facing. By acquiring Sun Microsystems in January 2008, Oracle—the owner of the most successful proprietary database in the world—acquired ownership of the trademark, copyright, and significant developmental resources of the most widely used relational database in the world, MySQL. This gave it visibility into, and to some extent ownership of, the open source relational database market that had the theoretical ability to impact its flagship database project. More recently, the company has been aggressively bolstering its cloud portfolio inorganically via acquisition. Since October of 2011, the list of companies Oracle has acquired for their cloud or SaaS-related business model includes the following:

- BigMachines
- BlueKai
- ClearTrial
- Collective Intellect
- Compendium
- Corente
- DataRaker
- Eloqua

- Instantis
- Involver
- Nimbula
- RightNow Technologies
- SelectMinds
- Skire
- Taleo
- Tekelec
- Vitrue
- Xsigo Systems

If acquisition patterns can be assumed to be a manifestation of strategy, the narrative emerging from Oracle is quite clear: cloud and SaaS are where the dollars are going, and Oracle needs to get there as quickly as possible.

Salesforce

Salesforce's toll-free contact number today is 1-800-667-6389. One potential alphabetical translation of that—the one the company features prominently on its website, in fact—is 1-800-NO-SOFTWARE. For a decade now, Salesforce has been investing substantial portions of its marketing revenue toward campaigns, conferences, and messaging about the idea of "no software." Taken literally, this assertion is absurd. Without software, of course, there is no Salesforce: it's not as if the company makes a tangible, physical product. Based on adoption rates of the service, however, and attendance at their conferences, the average customer seems perfectly fine with a message of no software.

What Salesforce is capitalizing on with this campaign, in part, is the appalling failure rates of on premise implementations of Customer Relationship Management (CRM) software packages. In a piece for ZDNet from 2009, Michael Krigsman collected these varying analyst estimates for the percentage of failed CRM projects:

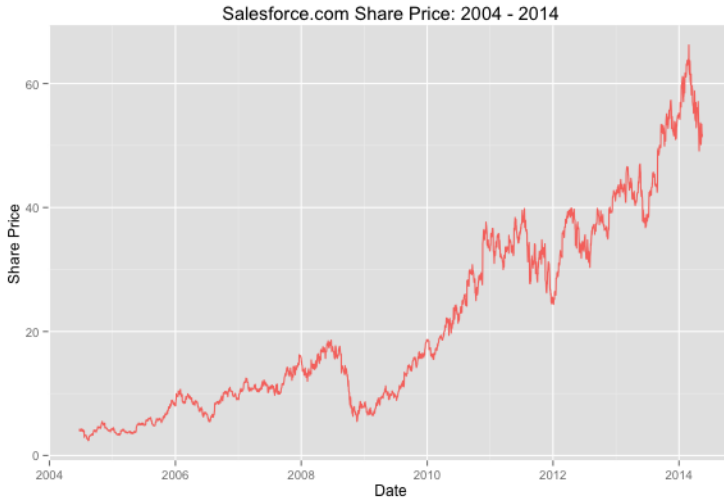
- 2001 Gartner Group: 50%
- 2002 Butler Group: 70%
- 2002 Selling Power, CSO Forum: 69.3%

- 2005 AMR Research: 18%
- 2006 AMR Research: 31%
- 2007 AMR Research: 29%
- 2007 Economist Intelligence Unit: 56%
- 2009 Forrester Research: 47%

Whichever survey you choose, something close to every other CRM customer failed to implement their software successfully. Definitions of failure vary, clearly, and undoubtedly some functional implementations were considered failures for what might be non-issues for other customers. Still, it's not difficult to understand why Salesforce would choose to go to market with a message of "no software": CRM software was not then, nor is it now, popular.

What if, however, there was no software to implement, just a service to consume? Pedants might argue with the semantics of the terminology, but it cannot be argued that there is a material difference between traditional shrink-wrapped CRM software and the browser-based alternative Salesforce and other properties are built upon.

How has this model performed over time? From a bottom line perspective, it's consistently been a money loser. In each of the last three financial years, Salesforce generated no profits, and its losses actually doubled in fiscal year 2014. Based on these facts alone, it would be reasonable to expect the market to have punished Salesforce. Instead, it has consistently rewarded it. Consider the following chart:



What possible explanation is there for the underperformance in profit and outperformance in share price? It's difficult to infer market intent with any degree of precision, of course, but it seems reasonable to assume that investors are simply applying a different set of expectations to Salesforce than to the wider market. Much as is the case with Amazon.

It is often said, with a great deal of justification, that the market is tremendously shortsighted. This lack of patience, in turn, manifests itself as companies who manage themselves from quarter to quarter and thus have difficulty occasionally taking one step back in order to take two forward. This is, in many respects, a foundation of Clayton Christensen's Innovator's Dilemma. Microsoft might have been best served by cannibalizing its own operating system business, for example, to competitively respond to the disruptions from the public cloud. This is enormously difficult to do in practice, however, because it runs counter to the markets' expectation for consistent profit and reward.

In both Amazon and Salesforce, however, the market appears to be taking a longer view. Recognizing that the addressable market for each is in part a function of scale, investors have to date been willing to trade current profit for revenue growth and capital investments aimed at building for the longer term. How long this will continue is unclear, but it's a multi-year trend at this point. Fred Wilson of Union Square Ventures discusses the company's strategy from an investor's perspective as follows:

> *The lesson here is that you can't just value a company by taking its current performance into account. You really need to have a view towards its future performance. And you need to understand why the company is not currently profitable....*
>
> *In the case of Salesforce...they are making huge investments in sales and marketing to secure additional customers. They are also making significant annual investments in R&D to maintain the market leadership of their existing products and bring new ones to market. If you think that Salesforce...can continue to grow their revenues at or near their current growth rates, then you ignore the current P&L and think about what a future P&L might look like.*
>
> **— FRED WILSON**

This special treatment is an important and under-appreciated competitive advantage for the company. As IBM's senior vice president, Steve Mills, said in an interview with GigaOm's Derrick Harris, "We don't have permission to not make money. We're not Amazon." Setting the implied sarcasm aside, this is an important point.

Responding to the Software Paradox is, for many companies, likely to be an expensive proposition. Developing software is an expensive business to be in, but developing it and delivering it as a service is that much more so.

Those companies that have market permission to invest in the necessary operational scale at present have an important advantage over those that are slaves to quarterly results, because sacrificing profit now for opportunity later may hold big dividends as the market for software as a standalone entity continues to decline.

VMware/Pivotal

It was 2006 when Amazon launched what would later come to be known as the Infrastructure as a Service (IaaS) market. IaaS made available in an on-demand fashion the basic building blocks for a wide range of workloads in compute and storage. While IaaS offered users fine grained control of their infrastructure, however, it did little to offload the operational workload from users. Perceiving an opportunity for customers who were willing to trade control for the convenience of offloading management tasks, however, Salesforce launched its Force.com PaaS platform a year after EC2, and Google followed seven months later with App Engine. Both of these products were aimed at users who wanted to deploy applications without worrying about the operational details. For all of their promise of lower

operational overhead, the products failed to see the traction that many expected. Particularly when measured against the meteoric rise of Amazon Web Services, growth within the PaaS category was anemic.

Not everyone had given up on the category, however. Then CEO of VMware Paul Maritz hired Mark Lucovsky away from Google and put him together with Derek Collison and Vadim Spivak to build, in Lucovsky's words, "something in cloud, for developers." The end result of these efforts ended up being Cloud Foundry, software intended to offer many of the same features that Google App Engine and Force.com offered.

When launching Cloud Foundry—what Maritz has termed "the 21st-century equivalent of Linux"—VMware made two important operational decisions. The first and most obvious was to release the project as open source, specifically under the permissive Apache license. This is an important decision in the abstract, but even more so given the context that it was created by VMware, a company whose financial fortunes were and are largely dependent on the sale of closed source, proprietary software. According to a November 2011 interview in *Wired*, this decision was driven by Collison and Lucovsky, and the ramifications are interesting to consider.

The biggest impact of the decision to open source the software, and later create an independent foundation around the asset, is that erstwhile competitors like IBM and VMware can collaborate with one another on the project. Reducing the friction associated with adoption of Cloud Foundry, meanwhile, has unquestionably fueled its rapid growth.

Less heralded than its open source availability, however, was the decision to make available a hosted version of the product from day one. Understanding, perhaps, that merely making the source code available is increasingly insufficient in a world in which the cloud has set an expectation of near-instant provisioning, VMware and subsequently Pivotal chose to invest in the infrastructure necessary to at least tinker with the software, if not host production applications.

It is a statement indeed when one of the largest providers of proprietary software creates a piece of software it compares in significance to the Linux kernel and chooses not only to release it as open source, but to work with competitors to improve it. To recognize that software's commercial value may have changed from the principles the company was founded on, and to act on this, is impressive. To go one step further and combine the software with services to ease adoption is even more so. Both are suggestive of an organization that understands the Software Paradox, and is actively adjusting to it.

What to Do

The Software Paradox and Your Business

Having seen the impact of the Software Paradox across a wide variety of organizations, from consumer to enterprise, startup to industry bellwethers, the obvious question is how to respond. There is no one path forward, as the appropriate organizational response will depend on a number of variables, including the resources on hand, current business model, market permission to and accessibility of adjacent markets, and so on.

There are, however, three recommended strategic considerations for organizations subject to the Software Paradox moving forward.

#1: ASSUME THE SOFTWARE PARADOX TO BE TRUE

The first step to solving any problem is to acknowledge the problem, which in this case means accepting that the upfront, realizable commercial value of software is in a period of decline. Even if you work for a Palantir or a Splunk and your business is currently an exception to this trend, it's useful to model the impact if only as a thought exercise. Many businesses, including successful ones, have been caught unprepared by unanticipated downturns in their ability to monetize software. As in the Innovator's Dilemma, few have predicted this in advance based on the mechanics of their existing businesses. More problematically, the more successful their history of generating software revenue, the more difficulty they have in envisioning challenges to it moving forward.

As a result, the most important recommendation for organizations of all shapes and sizes moving forward is to anticipate worst case scenarios at a minimum. Even in cases where organizations cannot or will not make some of the operational changes recommended below, the exercise of focusing on nonsoftware areas of a given business can help identify under-realized or -appreciated assets within an organization. Particularly ones for whom the sale of software has been low effort, brainstorming about other potential revenue opportunities is unlikely to be time wasted.

One vendor in the business intelligence and analytics space has privately acknowledged doing just this; based on current research and projecting current trends forward, it is in the process of building out a 10-year plan over which it assumes that the upfront licensing model will gradually approach zero revenue. In its place, the vendor plans to build out subscription and data-based revenue streams. Even if the plan ultimately proves to be unnecessary, the exercise has been enormously useful internally for the insight gained into its business.

#2: IDENTIFY AND PURSUE VALUE

Whatever the outcome of the previous exercise, it is important for every business to be continually moving in the direction of value. In an industry that prides itself on speed of innovation and change, and whose history demonstrates same, it's counterintuitive that the conventional wisdom is that current market success equates to future market success. But this is, for better or for worse, typical. It is important to actively counteract this dangerous contentment with the status quo by continually evaluating the actual value of a market, whether that's by way of quantitative metrics like margin or more qualitative assessments of untapped opportunity or inbound risk.

Quantitative metrics will be of most use in existing businesses, where comprehension of a given market is high. They will, however, be less efficient in new markets. The best example of this is perhaps Apple's experience in the tablet space. The company's assessments of the opportunity for Newton-like devices were, in hindsight, clearly overly optimistic, to the extent the company drove itself dangerously close to the breaking point. Almost the exact same product space would prove fantastically lucrative a decade later; the iPad revenue stream alone is a Fortune 500–type business. In a best case scenario, businesses will continually monitor, both quantitatively and qualitatively, their individual product lines for signs of a decline and have plans in place to react when it does arrive.

The case study for this behavior within the technology industry is IBM. Steve Mills, IBM's senior software executive, in fact, regularly describes "moving toward value" as a key component of the company's overall strategy—an assertion that is born out, in fact, by the company's regular decommitments from underperforming markets like PCs or x86 servers, markets that may still be profitable but unable to yield the types of margins the company prefers.

However value is assessed, ultimately, organizations need to be prepared to move toward it. If disruption has not come to your software market yet, it is on the

way. And as the saying goes, if you don't find yourself a seat at the table post-disruption, you will be the meal.

#3: DIVERSIFY THE BUSINESS

Wherever possible, it's useful for businesses to hedge themselves against the potential for declines in their business. Importantly, this does not involve the abandonment or deprecation of existing software revenue lines. Quite the contrary: these should be maximized for as long as may be sustained. Diversification of revenue sources, however, is a long-proven strategy for unexpected disruptions to one or more lines of business. Software-dependent organizations, therefore, should be actively working to identify adjacent or emerging revenue opportunities that could complement or even outperform their existing software businesses. The most common pattern of model expansion, in fact, will be organizations using their software margins to effectively subsidize the generation of the business models that will complement it in a best case scenario, or replace it in a worst.

The highest profile example of this in practice today may be Microsoft. Even as it was relentlessly generating revenue by way of its flagship software offerings, it was pouring money into its own cloud infrastructure. According to the company, it has spent $15 billion on its cloud infrastructure to date, with no signs of the investments slowing. This is an enormous expense, particularly relative to the costs of developing software, but it is the scale that's necessary to be competitive in this market: Google spent $2.35 billion in the first quarter of 2014 alone according to its financials. But if Microsoft can efficiently generate revenue using the lower expense model of software, why would it feel compelled to spend so freely to compete in the services world? The only logical explanation for the level of commitment is that the company has projected or at least anticipates the possibility of disruptions to its core revenue streams, and is diversifying the business ahead of these challenges.

It should be noted that this is a good practice even if one finds the evidence suggesting a broad-based decline in commercial software businesses unpersuasive. The fact is that the majority of software businesses today are leaving money on the table by focusing strictly on the production and delivery of software at the expense of other customer needs in the process, whether that's operational assistance (services), improved decision-making (telemetry analytics), or the ability to amortize their capital outlay over longer periods of time (subscription models). Irrespective of what software organization leaders might think of the long-term forecast for software as a revenue-generating asset, it is irresponsible not to pursue additional

avenues of growth for the business, or to not attempt to protect the organization by diversifying its revenue-generating abilities.

Alternative Models to Explore

Beyond the above exercise, which requires detailed consideration of abstract principles and their precise relation to your business, what are some specific models to explore that can act to mitigate any decline in software-related revenues while opening up net new lines of business? There are too many to detail in these pages, so obvious candidates such as advertising-supported business models are omitted here. The following are business models that every producer of software, be they an organization of hundreds of thousands of people or a two-person startup, should consider.

SHIFT TO SUBSCRIPTION LICENSING

The simplest transition for many who would sell software, from a logistical standpoint if not public relations, is to transition customers to subscription models. For enterprises, this is already typical for support and maintenance, and in many cases licensing. At Red Hat, for example, 87% of the company's revenue is subscription-based. Even in the consumer world it's not without precedent.

As common as the model might be, the Adobe case clearly demonstrates, users may resist the idea of a subscription. There's no getting around the fact that there is, at least on a consumer level, some discomfort with the idea of renting rather than owning software. Some even compare the practice to sharecropping. Nor is this attitude entirely without justification. Most obviously, renters can have software taken away from them, while buyers at least have access to the version they purchased guaranteed. More important, customers will tend to pay more over the longer term for subscription software versus that which is purchased up front. The delta varies depending on type and category, but in general, the reason businesses shift to annuity-style payments versus upfront windfalls is that they have greater longer-term value.

But the fact is that a variety of markets are trending toward rentals. Today, millions of users all over the world forgo purchasing music in favor of monthly subscription fees to large catalogs such Pandora, Rdio, or Spotify. Millions more have given up the purchase of DVD or Blu-Ray discs in favor of online media streamed by Amazon or Netflix. Virtually every commercial SaaS application consumer or enterprise is purchased via subscription. Even in the mobile world, hundreds of apps have abandoned upfront pricing for software in favor of subscription-

like in-app purchases. Even better, from a user's perspective, is how the software subscription model incents a different model of development. Traditionally, software manufacturers are forced to choose between improving a particular version and holding new features back to improve their chances of persuading consumers to upgrade to the next version of the software. Under a subscription model, a customer's desire for more up-to-date software with the latest features is perfectly aligned with the vendor's need to minimize churn by continually demonstrating value. Even in cases such as Microsoft Office, where additional features may be little or no incentive for subscription, integrations with backend services can make up the difference.

The net for businesses that continue to charge an upfront, one-time licensing fee is that you should at least evaluate the possibility of transitioning customers to monthly subscription models. Whether it's enterprises increasingly paying for their infrastructure on a monthly basis to consumers increasingly buying their media—or in some cases software—the trajectory of payment models is clear. Software may be more difficult to sell, but it's generally a better and more viable proposition when sold over time.

OFFER YOUR SOFTWARE AS A SERVICE

If a transition from upfront licensing to a subscription model involves the least amount of organizational effort, embracing a SaaS model may involve the most. Most pure-play software organizations today have some operational infrastructure competency, even if it's just for build and test purposes. But very few—even among larger, well-resourced players—have the current ability to create a production-quality hosted version of their product. In a historically tight hiring environment, after all, it can be difficult enough to hire the engineers necessary to develop the software; finding those with the operational skills to host it, as well as to design the requisite billing, account management, etc., pieces that transform it into a SaaS offering is exponentially more so.

Unfortunately, in spite of these difficulties, hosting a given piece of software is becoming necessary in an increasing number of categories. More often than not today, availability and convenience will trump features and performance. Much as MySQL once enjoyed an adoption advantage over PostgreSQL simply by virtue of being the only one of the two available in the Linux repositories, so too today does software accessible as a service have an advantage over that which must be downloaded, installed, and configured—even if the latter is open source. In cases where it's not practical or possible to host the software for production, offering a trial,

sandboxed environment like Cloudera Live can be an enormously useful recruitment tool. Another avenue to monetize services is hosting a complementary service such as MongoDB's Monitoring and Management service; such value-add services can be an excellent blend of software and service-based models.

It is also worth noting that sustaining a service-oriented software offering does enjoy some advantages over traditional distributed models. As Andreessen Horowitz's Preethi Kasireddy and Scott Kupor describe, development and support costs can be substantially lower for SaaS businesses.

> *In a perpetual license business, the R&D (and support) teams are often maintaining multiple versions of the software, with multiple versions running in the wild. Even Microsoft had to finally—12 years later—deprecate its support for Windows XP, despite all sorts of customers from ATM operators to federal, local, and international governments mourning the loss.*
>
> *This generally doesn't happen in SaaS because all customers are running on the same hosted version of the software: one version to maintain, one version to upgrade, one version on which to fix bugs, and one physical environment (storage, networking, etc.) to support. Given that software companies at maturity often spend 12–15% of their revenue in R&D, this cost advantage is very significant and further enables SaaS companies to be even more profitable at scale—particularly if they use multi-tenant architectures. Not to mention that this simplified hosting and support model is the very linchpin for long-term SaaS customer success and retention, especially as compared to the buy-but-don't-use "shelfware" behavior that characterizes perpetually licensed enterprise software.*
>
> **— PREETHI KASIREDDY AND SCOTT KUPOR**

The costs and challenges notwithstanding, the future is services. Perhaps the best example of the industry's march in this direction is the public cloud. In almost every case, a physical server will outperform the virtual equivalent offered up by public clouds. And yet the adoption of public cloud has been sufficient to force Dell to go private, IBM to decommit from the x86 server market entirely, and HP to try and charge for firmware upgrades. This is the power of convenience. Much like the camera you have with you being better than the high-end SLR that's too heavy to carry around, developers—the new kingmakers within the enterprise—are heavily advantaging time to productivity when it comes to technology selection.

Which means that software providers need to adapt to a market that isn't just evaluating the capabilities of their offering, but how quickly they can be spun up. Screaming performance and differentiated features are wonderful, but as technologies from MySQL to MongoDB have amply demonstrated, they are far from the end all and be all. The most dangerous belief for any software company today is that the solution to their adoption problem lies in better software engineering.

The solution to problems of adoption is not a better product, but a focus on barriers to adoption. Which in many cases means offering the software as a service, daunting as that task may seem. Organizations with the ability to both develop and host their software will be far more insulated from any software-related revenue declines than pure-play competitors, which is why it's useful for every software organization to at least talk about the possibility of developing the capability internally or tightly partnering externally.

BUILD AROUND DATA

For many years, as Basecamp's Jason Fried reminds us, lumber companies treated sawdust, the byproduct of their operations, as industrial waste. Worse, the waste was a hazard. Besides being highly flammable and thus a potential cause for fire or explosion, sawdust is a known carcinogen, bacterial vector, and can have detrimental effects on local water systems. Not surprisingly, then, lumber mills had little love for sawdust. At least until they learned that they could sell it.

In searching for a use for the scrap wood left over from one of his factories, Henry Ford decided to use it in the manufacture of charcoal briquettes, which the subsequent Ford Charcoal Briquettes company did from 1921 until it was sold to the Kingsford Chemical Company in 1951. But charcoal was just one use for sawdust.

> *The lumber industry sells what used to be waste—sawdust, chips, and shredded wood—for a pretty profit. Today you'll find these by-products in synthetic fireplace logs, concrete, ice strengtheners, mulch, particle board, fuel, livestock and pet bedding, winter road traction, weed killing, and more.*
>
> **— JASON FRIED**

The software equivalent of sawdust today is data. Every second a piece of software runs, every time it's deployed, every time a user interacts with it, every time a transaction is completed, interesting and potentially valuable data is generated. Today, a small number of companies are leveraging this data in any sort of sys-

tematic, meaningful way outside of categories such as web analytics, where the practice is common.

The innate appetite for this information, however, is immense. One of the operating principles behind the success of wearable fitness platforms like the Fitbit or the Jawbone One is the Hawthorne Effect. Coined during a study of manufacturing worker productivity, it simply suggests that humans perform better when they know they are being monitored. Today we can see the implications of this as companies are able to compare themselves against a baseline of other users, as in New Relic's Application Speed Index, which allows a given customer the chance to compare their performance to other similar customers in an anonymized fashion.

Data-based revenue models are certainly not new; Acxiom, Bloomberg, Fair Isaac, Lexis-Nexis, and others have built large revenue streams off of controlled, borderline monopoly–level access to data streams. Today, however, data is everywhere, which means that the opportunities to monetize have multiplied exponentially.

Perhaps the most attractive feature of data-based business models, however, is the degree to which they can function as a moat or barrier to entry around your business. This is a lesson that Apple inadvertently taught the industry during the launch of its Maps application. Aesthetically, and it can be argued functionally, Apple's Maps software eclipsed Google's offering virtually overnight. Unfortunately for Apple, mapping applications are dependent on the corpus of data behind it, and Google's was and is substantially superior. While it was possible, then, for Apple to make up ground in software very quickly, doing so in the world of data was substantially more challenging even for a company of its resources. There are no shortcuts, as data simply cannot be generated overnight. There are only two means of producing it: collecting it over time, or acquiring an entity who has done so. Unlike software, then, which is a thin shield against would-be market entrants, organizations that amass a large body of data from which to extract value for themselves or their customers are well protected against even the largest market incumbents.

Every software organization today should be aggregating data, because customers are demanding it. Consider, for example, online media services such as Netflix or Pandora. Their ability to improve their recommendations to customers for movies or music depends in turn on the data they've collected from other customers. This data, over time, becomes far more difficult to compete with than mere software. Which likely explains why Netflix is willing to open source the majority of its software portfolio but guards the API access to its user data closely. Over in

the enterprise world, Cloudera is using its own Hadoop infrastructure to aggregate customer data to inform its own support approach, and in the consumer electronics space, Nest expects revenue from its data-oriented utility provider business to eventually eclipse the sales of its primary product, the Nest thermostat.

Even for businesses that lack a cohesive plan for using their data, the resources to really put it to work, or both, it is imperative to at least begin collecting that data as soon as possible. It is always possible to create a plan and the software to execute it later. Data not collected, however, cannot be conjured on a whim.

THINK ABOUT YOUR SOFTWARE AS AN ASSET, NOT MONEY

The primary difficulty for many software producers, particularly those that have experienced a great deal of commercial success, is that they begin to lose the ability to differentiate between software and revenue. History, of course, demonstrates conclusively that this is a problematic approach. Software that 10 years ago would have had a seven-figure price attached to it is today available for free as open source. Certainly there remain areas where software commands a very high price, but the number of these opportunities is smaller by the year as the portfolio of open source solutions improves in both quality and volume.

In such a climate, the more appropriate way to think of software is as an organizational asset: nothing more and nothing less. Looking at software without assuming monetization can allow more strategic opportunities to emerge.

In spite of the acquisition cost of OTI, for example, and an additional $40 million invested in the platform, IBM made the decision to open source the Eclipse platform, at once making it more difficult to monetize and available to competitors. Why would they take such a risk? Because the perceived benefits, from a broader, more stable community to increased pricing pressure on a competitive product, Microsoft's Visual Studio, outweighed the costs. This step was only possible, however, because the company considered the software an asset to be leveraged, as opposed to revenue incarnate.

Why would Google, for its part, openly publish details of its MapReduce apparatus and the Google filesystem, which Doug Cutting and Mike Cafarella would later use to create Hadoop? Because one of the biggest challenges to software organizations is hiring. In a world in which Google had kept details of MapReduce and the Google filesystem private, it would be impossible for them to assess these skills in external candidates. Worse, each new hire would have to be exposed and on-boarded to a very different programmatic approach. By thinking about software less as something to be protected, then, Google was able to publish details that had

a chance to dramatically improve the efficiency of its recruitment and training, which collectively would easily offset the cost of giving its competitors insights into innovative internally developed technologies.

The key realization for any organization, then, is to not elevate software to an untouchable status. If every business has three types of customers—those that will pay, those that might pay, and those that will never pay—it's possible to use software to extract real value even out of customers that will never in fact be paying customers. Software can be used for direct monetary gain, to be sure, but used strategically it can accomplish things money could never buy. When it comes to the value of software, then, remember to keep an open mind.

FULL STACK STARTUP

More relevant to smaller businesses than larger entities, generally, the full stack startup was mentioned previously in the context of the Nest case study. The idea is similar to classic vertical integration but more narrow in scope. Whereas classic vertical integration stories such as automotive manufacturing extend deeply into supplier territory, such as Ford manufacturing its own steel, full stack startups are those whose focus extends to each layer necessary to deliver the desired user experience. Their equivalent of manufacturing steel—owning and maintaining the underlying technical infrastructure—may have no bearing on their ability to target the opportunity, and as such, many full stack startups are content to effectively outsource their infrastructure to public cloud suppliers or other infrastructure specialists.

But realizing that the experience will be shaped by factors beyond just the software, full stack startups build or acquire competencies in all areas necessary to shape the user experience. The disadvantages of the process are primarily effort and cost centered. While the costs of developing software have plummeted in recent years thanks to a combination of open source software, public cloud infrastructure, and free or low-cost SaaS applications, the same is not true of nondigital startups. As James Park, CEO of Fitbit, told the *Wall Street Journal*:

> *If you are releasing software, you can do multiple deployments, and constantly tweak it. With hardware, you make your bet a year-and-a-half in advance, then you live with it. Mistakes can be expensive. Nowadays things are easier, because of Kickstarter and things like that. This is a capital-intensive business. I would tell others to maximize things like crowdfunding.*
>
> **— JAMES PARK**

These costs notwithstanding, depending on the area of opportunity, a full stack startup might be the only realistic approach to a given market. As Dixon said when he coined the term:

> *Prominent examples of this "full stack" approach include Tesla, Warby Parker, Uber, Harry's, Nest, Buzzfeed, and Netflix. Most of these companies had "partial stack" antecedents that either failed or ended up being relatively small businesses.*
>
> **— CHRIS DIXON**

It's difficult to conceive of how companies like Nest, Tesla, or Uber could have achieved what they have, had they taken a software-only approach to a given market. In some cases, such as Netflix, it's difficult to imagine them existing at all absent this approach—imagine if the company had to wait for studios to license its technology to stream their media.

None of which is to say that the full stack approach is going to be the correct one in every setting, just that it's an important question to ask as software strategies are shaped. It's even possible for a full stack approach to graduate to true vertical integration, as in the case of Apple. Apple has long been an adherent to the full stack philosophy, delivering a tightly integrated experience that it controlled top to bottom, even if it outsourced the actual manufacturing. As it moved into its own chip manufacturing late in the last decade, however, it extended that philosophy even further into true vertical integration territory.

The most important consideration, integration semantics aside, is to determine what a business needs to control to deliver value to a customer. It is from there that everything else, strategy included, follows. Software may be the most important given component, but if it's one of many, the wider strategy needs to take that into account.

Final Thoughts

"The measure of intelligence is the ability to change."

— ALBERT EINSTEIN

According to *Bloomberg Businessweek*, since the year 2000, an information technology company was counted among the world's five largest businesses every year but two, 2007 and 2008. Microsoft was by far the most successful, serving as the industry's representative on that list eight years out of the decade beginning in 2001. It hasn't made the list since 2010, however, even as 2013's list included both Apple and Google. This changing of the guard perfectly symbolizes the transition currently underway in the industry, one leading away from software as revenue and toward revenue using software.

It is a paradox that the economic value of software is falling even as its strategic value rises, and paradoxes are by definition challenging to accept. But look no further than Microsoft's absence for confirmation of the risks. Even as the business continues to print money with its two most popular software franchises, it is retooling itself to compete in a very different landscape, and its new leadership reflects that.

Because the software industry has generated so much wealth historically, because it continues to today, and because software really is eating the world, it can be difficult to accept the idea that its intrinsic commercial value is in decline. But the evidence is both broad and conclusive. When large, successful incumbents are having difficulty growing license volume, margins, revenue, or all of the above, and new emerging players are releasing as open source assets that would have been worth millions a decade ago, it's safe to say that a new pattern is emerging.

Software has never been more important than it is today, but software producers expecting to match the performance of years past are setting themselves up for disappointment. There are exceptions, but in the majority of cases, the realizable revenue and margins of traditional standalone software businesses are trending downward, and there is no reason to expect a recovery. From startups to big busi-

nesses, enterprise to consumer, it's simply getting harder for businesses to make money selling software by itself.

The silver lining is that the slope of the decline is mild, which means that there is time to adapt—assuming organizations can acknowledge that there is a problem in the first place. Open source and the rise of the developer kingmaker have altered procurement fundamentally and permanently, but enterprise buyers at least have three decades of conditioning telling them that they must pay for software. Many buyers, frankly, will keep paying for software not because they have to but simply because it's routine. Intelligent, adaptive organizations will therefore use whatever software runway they have left to subsidize the generation of new complementary or even replacement lines of revenue. Their less-progressive competitors, meanwhile, will be left to fight over a budget pool that will grow smaller every year.

Once upon a time, an entire industry *knew* that the economic value wasn't in software, when in fact it was. Today, we *know* the economic value is in software licensing, when in fact it increasingly is not. With history unequivocal on the outcomes for those who *know* what the value is versus those willing to question it, everyone producing software should be considering what the Software Paradox means to them.

About the Author

Stephen O'Grady is a cofounder of the developer-focused technology analyst firm, RedMonk. Regularly cited in publications such as the *New York Times*, *BusinessWeek*, and the *Wall Street Journal*, Stephen's work revolves around understanding developer needs and trends and working with businesses to help them work more effectively with the New Kingmakers. Although his birth certificate says New York City, Stephen is a Red Sox fan, born and raised. A graduate of Williams College, Stephen lives in midcoast Maine with his wife.