Format Obsolescence: Assessing the Threat and the Defenses

David S. H. Rosenthal
Stanford University Libraries, CA

Abstract

Much of the work in digital preservation has focused on the perceived threat of format obsolescence. The standard approach combines tools to validate formats and collect format metadata, registries preserving format specifications, and format obsolescence notification systems. The idea is that when a format becomes obsolete a notification will be issued, a converter created from the specification in the registry, documents in the format identified by means of the preserved format metadata, and converted using the converter into a not-yet-obsolete format.

After a decade and a half of work based on this model it is time to ask these questions:

- Are current formats becoming obsolete?
- If a current format becomes obsolete, how likely is this approach to succeed in keeping documents readable?
- Are there alternative approaches that would cost less and be at least as likely to succeed?

1 Introduction

Fifteen years ago, Jeff Rothenberg's seminal article in *Scientific American* entitled "Ensuring the Longevity of Digital Documents" drew public attention to the problem of format obsolescence:

> "... digital documents are evolving so rapidly that shifts in the forms of documents must inevitably arise. New forms do not necessarily subsume their predecessors or provide compatibility with previous formats." [24]

At the time he had many horror stories to draw on, for example the fate of the Domesday Book application for the BBC Micro computer [6].

Subsequent work in digital preservation focused largely on combating this perceived threat of format obsolescence, developing a standard approach which combines tools to validate formats and collect format metadata [12, 27, 15, 18], registries preserving format specifications [28, 26], and format obsolescence notification systems. The idea is that when a format becomes obsolete a notification will be issued, a converter created from the specification in the registry, documents in the format identified by means of the preserved format metadata, and converted using the converter into a not-yet-obsolete format.

Despite all the effort and resources invested in this standard approach, examples are lacking of content that has been rescued from format obsolescence in this way. Indeed, when challenged, proponents of this approach have failed to identify even one format in wide use when Rothenberg wrote that has gone obsolete in the intervening decade and a half. If format obsolescence were a significant threat to the longevity of digital documents one would have expected at least one outbreak in fifteen years.

First, this paper asks why, if it was so plausible for Rothenberg to raise the spectre of format obsolescence, it has failed to appear. Second, it assumes that despite this analysis format obsolescence does appear, and examines the practical details of the standard approach to assess how likely it is to succeed in rescuing documents in the doomed format. Third, it looks at an alternative approach to assess whether it would be more cost-effective.

2 Format Obsolescence

Rothenberg bases his explanation of the threats to the longevity of digital documents on the example of a CD, containing documents describing the location of his hypothetical fortune, discovered fifty years later by his hypothetical descendants. The context makes it clear that the way the documents got on to the CD was via a *desktop publishing* system. In 1995, there were many exam-

ples of desktop publishing document formats going obsolete, but fifteen years later the current version of Open Office has support for reading and writing Microsoft formats back to Word 6 (1993), full support for reading WordPerfect formats back to version 6 (1993) and basic support back to version 4 (1986).

Rothenberg writes as if incompatibility were the result of inevitable natural processes, rather like random genetic mutation. But in fact it is the result of deliberate decisions by software developers, who must weigh the benefits it brings against the costs it imposes. For widely used infrastructure software the total cost, summed across the many users, almost always vastly outweighs any possible benefits, which typically accrue only to a few. In fact, developers have a derisive name for the process that re-implements existing functionality in an incompatible way; they call it "re-inventing the wheel".

There are two main reasons why format obsolescence in digital documents stopped around the time Rothenberg was writing; maturing markets and the Web.

2.1 Maturing Markets

The first reason lies in the nature of technology markets, and was explained by a book published the year before Rothenberg's article. In "Increasing Returns and Path Dependence in the Economy" [3] Brian Arthur explains that these markets exhibit increasing returns to scale, commonly referred to as *network effects*. An example is *Metcalfe's Law*; the value of a network goes as the square of the number of nodes. An immature technology market is characterized by a large number of competing products, whose market shares are small and variable as random factors affect their success. The market matures when these random factors increase the market share of one product enough for the network effects to take over; its market share then rapidly expands and it comes to dominate the market.

In immature markets, such as desktop publishing was before 1995, format obsolescence is common because the product concerned is not widely used enough for the costs to the small number of users to outweigh the benefits. As the market matures, the winning product has incentives to make it easy for users of any competitors with significant market share to migrate. Their formats do not go obsolete because they are supported by the winner. The winner's format does not go obsolete because it is widely enough used for the costs of obsolescence to dominate any possible benefits.

Further, once a dominant product has captured a market in this way the incentives for innovation invert. Before market capture, products are driven to innovate as fast as possible by the idea that each successive innovation will be the one that expands their market share enough for the network effects to kick in and drive it to market capture. After market capture, innovation becomes risky for the dominant product. It will innovate as little as possible consistent with retaining its dominant position. Indeed, a well-known and often employed strategy is for the dominant company to buy new, innovative companies that appear in or near its market to suppress their innovation and thus prevent them becoming a threat. Just as market capture increases the costs of incompatibility by multiplying the number of users who bear those costs, it reduces the benefits of incompatibility by reducing the number of potential new users that the benefits would attract.

Those who believe that format obsolescence is prevalent often point to the evolution of Microsoft's Office formats as an example of format obsolescence, but this argument demonstrates a misunderstanding. The rapid evolution of formats is another aspect of market capture. Office is such a dominant product that it cannot grow its market share. To make it increasingly profitable, Microsoft must persuade the customers who already have Office to pay for it again, through purchasing upgrades. An important technique for doing so is to ensure that, by default, the new version of Office writes a format that the previous version cannot read. Thus, once new computers with new versions of Office arrive, those with the old version will continually encounter documents they can't read without upgrading.

It is important to note that this is an example of format *innovation*, not format *obsolescence*. Obsolescence would occur only if the new version could not read files written by the old version. If this were the case those with the new version would want to downgrade, the opposite of the effect Microsoft intends. A recent example illustrates how difficult it now is for Microsoft to render formats obsolete. In 2008, Microsoft announced that a service pack for Office would remove support for some very old formats. The public response was so hostile that it took less than a week for this announcement to be rescinded [21].

Thus, the only formats that do go obsolete are those that failed to gain significant market share *before* the market matured. In the nature of things there is very little of value in these obsolete formats.

2.2 The Web

The second reason is a change in the nature of the documents themselves. The publishing medium for which desktop publishing systems were designed was paper. The design goal of their document format was therefore preserving the state of the application so that the document could later be re-opened by the same application for further manipulation. It wasn't transferring information to readers; that was intended to take place on paper. In 1995, the use of these formats for this transfer was an un-

intended by-product; it typically took place by exchange of floppy disks or via e-mail.

As Rothenberg's article appeared, the Web was in its infancy. Less than 6 months later Stanford's HighWire Press started the transition of academic journals to the Web by putting the *Journal of Biological Chemistry* online [10]. Less than a year later Netcraft started tracking the growth of the Web [16]. A bit more than a year later the Internet Archive started collecting web sites for posterity [11].

The relevance of the Web for this discussion is that the change from paper to the Web as the medium by which electronic documents were published meant that the goal of a document format was no longer that it be written and subsequently read by the *same* application, but rather that it be read by multiple *different* applications from the one that wrote it. Note that in this environment the costs are incurred not when a new, possibly incompatible, format is *added* to the Web browsers, something that happens all the time via browser "plugins", but only when support for an old format is *removed* from the Web browsers. The cost of retaining support for old formats in a Web browser is minimal, so there is no constituency advocating their removal.

The goal of publishing an electronic document is to transmit information from an author to readers. Once the assumption that the author and the readers all use the same application in the same environment is removed, the cost of incompatibility becomes almost infinite. The number of readers is, the author hopes, very large and the cost to each reader of being unable to read the document is also very large. Further, the benefits of incompatibility depend on the author being able to persuade the readers to upgrade their Web browsers not in order that they may read a new format, which they may easily do by installing a new plugin, but in order that they *not* be able to read an old format. This would be a tall order for any marketing department. Thus the switch from private to public, published documents again increases the costs and eliminates the benefits of format obsolescence.

2.3 Open Source

An independent argument leading to the same conclusion starts by asking whether we can construct a scenario in which a widely-used format becomes obsolete, i.e. documents in the format can no longer be rendered.

Almost all widely-used formats have at least one open-source renderer, and most have several. For each of these formats, the source code for the entire software stack, from the application such as Open Office (for the Microsoft Office formats) or ghostscript (for Adobe's PostScript and PDF), the Web browser, the operating system kernel, the BIOS and even a software emulation of the Intel hardware, together with all the tools needed to build a working environment, is written in ASCII, so is itself immune from format obsolescence.

Further, this source code is carefully preserved in a range of source code repositories, such as SourceForge. The developers of the various projects don't actually rely on the repositories; they themselves keep regular backups of the repository contents. All the tools needed to build a working software stack are preserved in the same way, and regularly exercised (most open source projects have automatic build and test processes that are run at least nightly).

As if this wasn't safe enough, in most cases there are multiple independent implementations of each layer of functionality in the stack. For example, at the kernel layer there are at least 5 independent open source implementations capable of supporting the entire stack[1]. As if even this wasn't safe enough, this entire stack can be built and run on many different CPU architectures[2]. Even if the entire base of Intel architecture systems stopped working overnight, in which case format obsolescence would be the least of our problems, this software stack would still be able to render the formats just as it always did, although on a much smaller total number of computers. In fact, almost all the Windows software would continue to run (albeit a bit slower) because there are open source emulations of the Intel hardware architecture.

The structure of open source's decentralized development model makes it very hard for incompatible changes to be adopted, and the history of the Unix-like environments demonstrates this. To take one example, in 2008 Kirk McKusick won the IEEE's Reynolds B. Johnson award for his 30-year stewardship of the Unix file system [29]. During that time he has overseen the growth of the file system code from 12K to 55K lines of code, and vast improvments in its performance and reliability. Through all these changes, over 30 years, as disks have grown bigger by a factor of nearly a million, there have been no incompatible changes in either the on-disk format or the application interface to the file system. In the unlikely event that a disk written 30 years ago had survived in working order, the current file system code could read it perfectly.

What is more, the source code is preserved in source code control systems, such as CVS and subversion. These systems ensure that the state of the software as it was at any point in the past can be reconstructed; not for the purposes of preservation but as an essential part of the normal software development process. Since all the code is handled this way, the exact state of the entire stack at the time that the content in question was rendered correctly can be recreated. Thus, even if incompatible changes had taken place in the open source

[1] Linux, FreeBSD, NetBSD, OpenBSD and Solaris

[2] NetBSD supports 16 of them

software stack, perhaps an incompatible change in the programming language [3] they would not prevent its use in rendering the content.

2.4 Closed Source

But what of the formats for which there is no open source renderer, only a closed-source binary plugin? Flash is the canonical example, but in fact there is an open source Flash player, it is just some years behind Adobe's current one. This is very irritating for partisans of open source, who are forced to use Adobe's plugin to view recent content, but it may not be critical for digital preservation. After all, if preservation needs an open source renderer it will, by definition, be many years after the original release of the new format. There will be time for the open source renderer to emerge.

But even if an open source renderer never emerges and only a binary plugin is available, and even if subsequent changes to the software into which the plugin plugs make it stop working, we have seen in the previous section that the entire software stack at a time when it was working can be recreated. So provided that the binary plugin itself survives, the content can still be rendered.

2.5 Summary

Clearly, there will always be some documents in formats that failed to gain significant market share before a market matured, and some markets take longer to mature than others. These arguments do not show that format obsolescence never happens. What they do show is that format obsolescence is not a significant threat to the overwhelming majority of digital content we wish to preserve.

They also allow us to characterize the kinds of format that are more at risk of obsolescence. These formats will be:

- In an immature market.
- Unsucessful.
- Proprietary.
- Without an open source renderer or a binary plugin for an open source environment.

3 The Standard Model

Although these factors make it unlikely that there will be a significant amount of content in a format that goes obsolete, we now assume that some format of this kind has gone obsolete and ask how likely it is that the current standard model of digital preservation will be effective in rescuing the small amount of possibly important content in it.

[3] These happen occasionally in the open source world, causing major controversy [**?**].

The standard model defines the following steps:

- Before obsolescence occurs, a digital format registry collects information about the target format, including a description of how content can be identified as being in the target format, and a specification of the target format from which a renderer can be created.
- Based on this information, format identification and verification tools are enhanced to allow them to extract format metadata from content in the target format, including the use of the format and the extent to which the content adheres to the format specification. This metadata is preserved with the content.
- The format registry regularly scans the computing environment to determine whether the formats it registers are obsolescent, and issues notifications.
- Upon receiving these notifications, preservation systems review their format metadata to determine whether they hold content in an obsolescent format.
- If they do, they commission an implementor to retrieve the relevant format specification from the format registry and use it to create a converter from the now-obsolescent target format to some less doomed format.
- The preservation systems then use this converter and their format metadata to convert the preserved content into the less doomed format.

It is interesting to note that the standard model is based on format migration, a technique of which Rothenberg's article disapproves:

> "Finally, [format migration] suffers from a fatal flaw. ... Shifts of this kind make it difficult or impossible to translate old documents into new standard forms." [24]

3.1 Format Registries

Given the characteristics of at-risk formats in Section 2.5, format registries face two main obstacles to fulfilling their part in the rescue of content:

- At-risk formats are unsuccessful, in immature markets, and so obscure that no binary plugin for an open source environment survives. This makes it unlikely that the format registries will hear about them at all.
- At-risk formats are proprietary, so proprietary that the open source community failed to obtain an adequate specification, or to reverse-engineer the closed format enough to create a renderer. Why would the registry will be able to obtain an adequate specification when others had failed to do so?

3.2 Format Metadata

Assuming that, despite these obstacles, the format registries did notice the target format and obtain enough information, the next steps are to enhance the format identification and verification tools, use them on the relevant content, and preserve the results.

Format identification typically does not require detailed understanding of the format, so that it is very plausible that enhancing these tools to play their part in the standard model would be feasible. For obscure, unsuccessful formats it is possible that detail beyond identifying the format itself, such as format versions or format options, might be lacking.

Format verification is more problematic. Building a useful format verification tool requires as much knowledge about the format as building a renderer. Paradoxically, in most cases this is *more* knowledge than is in the format specification. The reason is that, in the real world, documents rarely conform exactly to the specification for their format.

An essential design principle of the Internet was enunciated by the late Jon Postel in 1981. It is called Postel's Law or the Robustness Principle:

> "Be conservative in what you do; be liberal in what you accept from others." [19]

Digital preservation is necessarily on the "accept" side of this law.

It is well-known that much of the HTML on the Web fails the W3C validation tests. A 2001 study reportedly concluded that less than 1% of Web content is even valid SGML [1]. Portico applied the JHOVE validation system to about 9M PDF articles they received from 68 journal publishers and reported that it classified a substantial proportion as either "not well-formed" or "well-formed but not valid" [22].

General observance of Postel's Law ensures that this epidemic lack of conformance to standards does not impair the readability of the content. Web browsers, PDF parsers and so on take great pains to produce legible output from faulty input. This has a number of important consequences for digital preservation:

- The specification of a format does *not* contain enough information to construct a usable renderer for the format once it is obsolescent. It omits the essential techniques needed to render the substantial proportion of non-conforming content in the format. These techniques are encoded only in the source code of the renderers in use before obsolescence. *Nothing less than preserving the source code of these renderers will suffice*. If the source of a working renderer is available, it doesn't matter whether the format's specification has survived.
- The use of format validation tools as part of the ingest pipeline for digital preservation systems is either actively harmful, or it is wasted effort:
  - It is actively harmful if content that fails vaildation is rejected for preservation. It is overwhelmingly likely that the rejected content is in fact fully legible. Use of the validation tool has prevented preservation of content that otherwise would have been.
  - It is wasted effort if, as archives typically report, they run the validation tool but accept the faulty content anyway. This is what they would have done had they not run the tool at all.
- Format classification and validation tools are not magic, they are software just like the software that generates the non-conforming content. Thus they will inevitably make errors. Each of these inevitable errors has one of two possible results:
  - *Something bad happens*, such as legible content being rejected for preservation. That makes the error rate of the tool a very important number to be minimized at all costs. There do not appear to be any measurements of the error rate of these tools, nor any plans to make such measurements.
  - *Nothing bad happens*. That makes the tool irrelevant, since not using it can't be worse than using it and having it give wrong answers.

Thus the contrast between the Platonic ideal of format specifications and the messy but working real world impels us to two counter-intutive conclusions. In most cases format metadata is either irrelevant or harmful to digital preservation, and preserving their specification is not a means to legibility for obsolete formats.

3.3 Format Notification

The next step in the process is for the format registry, in one of their regular scans, to notice that the target format is obsolescent. Given that the target format is so obscure and proprietary that there is no open source renderer, nor any binary plugin for an open source environment, and very little content in it, the probability of a successful notification is very low. We have already assumed that despite these factors, the registry was successful in noticing the format and obtaining information about it. The registry still faces obstacles to a timely notification:

- Clearly, if a widely used format such as PDF were to become obsolete, the registry could hardly fail to notice it. But the formats whose obsolescence it

will be called upon to notice are little-used. They will end their lives in neglect and obscurity. How is the registry to detect that they are obsolete?

- It is likely that the obsolescence of the format will be discovered by some reader failing to render some long-forgotten content. How are they to know that the correct thing for them to do is to report this to the registry? Even if they know this, why are they motivated to do it? Reporting the problem will only result in the reader's desire for access being satisfied after a long delay, if ever.

3.4 Format Converters

On receiving notification from the registry, digital preservation systems are to review their format metadata to determine whether they hold content in the obsolescent format. If they do they are to commission an implementor to retrieve the specification for the format from the registry and create a renderer for it. How likely is this process to succeed?

The problem facing the implementor is analogous to, but in some respects more severe than, that of "clean room" implementations of languages and protocols. These are routine in the IT industry. The goal is to build a product compatible with a competitor's using only public information, such as published specifications[4].

Experience of this process reinforces the conclusion of Section 3.2 that the specification alone is inadequate. Implementors also need access to a working product, so that when they find aspects of the specification that are obscure or absent, they can experiment on the product to clarify its behavior. One aspect almost guaranteed to require this is the behavior of the renderer when faced with non-conforming content.

The assumption that the format is obsolescent means that the implementor does not have access to a working renderer. If a working renderer were available in either source or binary form, the preservation system could use it, there would be no need for a new one. The implementor in our case is thus faced with a significantly harder task that the "clean-room" implementor, needing to rediscover the appropriate behavior for non-conforming content and other obscure cases *ab initio*.

Experience with these "clean-room" implementations also allows us to estimate the resources required by this step of the standard model. For example, there are several open source implementations of the Microsoft Office formats. Perhaps the leader among them is Open Office, whose history [31] reveals a very large investment; it was originally developed as a commercial product, and its development continues to be subsidized by Sun Microsystems as the basis for a commercial product. To achieve its current functionality has taken a large, salaried team more than a decade, despite their task being eased by ready access to Microsoft Office and to other open source competitors. It is not credible to expect that even this level of effort could be justified by digital preservation activities alone, let alone the considerably greater effort that would have been required had they not had access to these resources.

Further, the digital preservation world often complains that even Open Office's level of fidelity is inadequate. Many of these criticisms are beside the point; they refer to inaccuracies in Open Office's rendering of the latest Microsoft Office formats. But from the perspective of digital preservation, the relevant criterion is Open Office's rendering of old, in fact obsolete, formats. After all, the precondition for the task of creating a "clean-room" renderer is that the formats are so obsolete that no functional renderer is available. Note that the most recent case of Microsoft Office format obsolescence was caused by Microsoft's deliberate decision to remove support for old formats [21]. This was so unpopular that it was immediately rescinded. No-one is arguing for Open Office to remove support for old formats, and it appears that even Microsoft's ability to do so has expired.

Many of the criticisms of Open Office's fidelity in rendering Microsoft Office documents relate to layout changes between the two renderings. These are beside the point for another reason. The changes are typically caused by small differences between the fonts available in Microsoft Office and in Open Office. They exist not because Open Office incorrectly interprets the Office document format, nor because the Open Office developers were incompetent. Fonts, and in particular the font spacing tables that drive the layout process, are protected by copyright. If the Open Office developers had copied the font spacing tables so exactly that there were no layout changes they may well have been breaking the law.

Just because a document format has gone obsolete does not mean that the fonts used by documents encoded in that format have gone out of copyright. The implementor is likely to face even worse intellectual property hurdles than the Open Office developers did. He will probably be faced with the orphan font problem; wanting to get permission to use a copyright font but being unable to find the copyright owner to ask for it. The need to preserve the fonts used by a document as well as the text motivates the ability of PDF to embed the fonts it uses into the document itself.

Behind this quest for pixel-perfect rendering lies an unrealistically black-and-white view of the world. Renderers are software. They all have flaws. Some are better

[4]The author has personal experience of this process, being a member of the team at Sun Microsystems that produced the first clone of Adobe's PostScript language from the published "Red Book" specification [8].

than others, but none is perfect. If we plot the quality achieved by a newly created renderer for a format against the cost of creating it we will get an S curve. A certain amount of money is needed to get to a barely functional renderer. Beyond that, quality increases rapidly at first, but after a while the law of diminishing returns sets in. Getting from 99% to 99.9% is very expensive; the cost of getting to 100% is infinite. Emulation of the entire original hardware and software environment is the only way to guarantee a pixel-perfect match with the original rendering. Anything else means that preserved content is likely to be rendered with even more flaws than the original.

The only real question is how much to spend to get to how close a rendering. The example of the Microsoft Office formats shows that the costs involved are likely to be far greater than access to the small amount of content preserved in an obsolete format can possibly justify.

3.5 Summary

Thus at every stage of the process it envisages, the standard model faces significant obstacles. Although it is likely that the model could succeed in rescuing content in a widely used, well documented, open format with acceptable if not perfect fidelity, it is very unlikely that it would ever be called upon to do so. For the formats that are at actual risk of obsolescence, the standard model is likely to fall at the very first hurdle, being unable to obtain adequate information about the target to perform the remaining steps. Even if all the obstacles could be overcome, the cost of doing so is likely to be much higher than could be justified by continuing access to the small amount of content in the obsolete format.

4 Alternate Model

Given that substantial fractions of both the R&D and operational budgets for digital preservation are devoted to aspects of the standard model, and given the unlikely prospect of these investments paying off in terms of actual rescued content, it is time to ask whether there is a more cost-effective alternative model.

In the search for a more cost-effective model three principles are important:

- Store only essential data.
- Perform only essential tasks.
- Delay performing tasks as long as possible.

4.1 Store Only Essential Data

One of the rare policies on which there is almost universal agreement among digital preservation systems is this, the *sine qua non* is that the original bits be preserved. Even systems that intend to perform format migrations (e.g. [25]) are reluctant afterwards to discard the original, pre-migration bits.

It is often said that "bit preservation is a solved problem", but at the scales and for the durations needed in digital preservation this is unfortunately not the case [20]. Multiple copies need to be stored to enable recovery from the inevitable failures. Determining how many copies are needed to achieve a specified probability of loss is an unsolved problem; the assumptions made to render the mathematics tractable are known not to be true in practice [4]. A conservative approach is to store more copies than indicated by studies such as Yano's [32] that use these assumptions. Equally often one hears that "storage is free" but, again, at the scale of real digital preservation and with an appropriate number of copies this is certainly not true. A long-term study of storage costs at the San Diego Supercomputer Center shows that raw media costs represent only about $1/3$ of total storage costs [14]. It is unrealistic to base the design of preservation systems on the idea that storage is currently free.

It is true that technological development in terms of increasing areal density (bits per unit square) has driven decades of exponential decrease in the cost per byte of raw disk media [7]. It is also true that technologies for future increases in areal density are in hand. But an industry expert has recently pointed out that the business case for ever-increasing capacity in low-cost 3.5" disk drives has become very weak [2]. The cost-per-byte curve may flatten in the near future for business rather than technology reasons; the CPU clock speed curve has flattened in the last few years for analogous reasons. It is unwise to base the design of preservation systems on the idea that storage will become free in the foreseeable future.

The question thus becomes what, in addition to the original bits, is it essential to store? In general, anything that is the result of running a program taking the original bits as input can be discarded, or at least does not need to be preserved with multiple copies. Examples include the (often voluminous) output of format identification and verification tools. Under what scenario would it suddenly become impossible to re-run these tools?

It can be argued that, at the scale of major preservation systems, running any program taking the entire preserved content as input is a major undertaking. But fixity checks necessarily require reading the entire preserved content on a regular basis; other tasks can be combined with the fixity checks so that the cost of reading the content once is amortized across all the tasks needing to do so.

4.2 Perform Only Essential Tasks

Which tasks, other than fixity checks, are essential to preservation? The analysis of Section 2 makes it likely that the vast majority of the content in a digital preservation system will never suffer format obsolescence in the

system's lifetime. Thus any effort invested in preparing that content for the eventuality will never generate a return. An example of such wasted effort is collecting more than minimal format metadata (such as Mime-Type) during ingest; besides the fact that it is more data to preserve that could easily be regenerated from the original bits, it is unlikely ever to assist in migrating the content in question because the need to do so is unlikely ever to arise.

4.3 Delay Tasks As Long As Possible

Delaying even the essential tasks until they are unavoidable has two important benefits:

- The time value of money means that even if the cost to perform a task remains constant, delaying it reduces its impact on the system's overall budget.
- In the information technology marketplace, the cost to perform a given task has historically dropped rapidly, often exponentially. At the same time, the quality with which the task can be performed has historically risen slowly as bugs were fixed and improved techniques implemented. These effects have normally been much larger than the interest rates used to compute the time value of money, greatly strengthening the case for delay.

In the small proportion of cases in which format migration becomes necessary, delaying performing it until the last possible moment, which is when the reader requests access, results in the migration being performed at the lowest possible cost and with the greatest possible fidelity. A technique for doing so in the Web environment, transparently to the reader, has been demonstrated [23].

4.4 Open Source

Although open source software has deep roots in computing history, in 1995 it was a small niche. Linux barely worked. The lawsuit between Unix Systems Labs and BSDi that freed the Unix source code was settled in 1993, but the details were still secret. Now, open source is a basic strategy for all but two of the big IT companies.

Historically, the open source community has developed rendering software for almost all proprietary formats that achieve wide use, if only after a significant delay. The Microsoft Office formats are a good example. Several sustained and well-funded efforts, including Open Office, have resulted in adequate, if not pixel-perfect, support for these formats. The Australian National Archives preservation strategy [9] uses these tools preemptively to migrate content from proprietary formats to open formats before preservation.

Even the formats which pose the greatest problems for preservation, those protected by DRM technology, typically have open source renderers, normally released within a year or two of the DRM-ed format's release. The legal status of a preservation strategy that used such software, or some software arguably covered by patents such as MP3 players, would be in doubt. But the efforts of the copyright owners to suppress these open source renderers pay tribute to their effectiveness; if they didn't render the content there would be no need to suppress them. Until the legal issues are clarified, no preservation system can make well-founded claims as to its ability to preserve these formats against format obsolescence. However, in most but not all cases these formats are supported by binary plugins for open source web browsers. If these binary plugins are preserved, we have seen that the software stack into which they plugged could be recreated in order to render content in that format.

It is safe to say that the software environment needed to support rendering of most current formats is preserved much better than the content being preserved in those formats (Section 2.3). If we ask "what would have to happen for these formats no longer to be renderable?" we are forced to invent implausible scenarios in which not just all the independent repositories holding the source code of the independent implementations of one layer of the stack were lost, but also all the backup copies of the source code at the various developers of all these projects, and also all the much larger number of copies of the binaries of this layer.

The all-or-nothing question that has dominated discussion of digital preservation has been how to deal with format obsolescence, whether by emulating the necessary software environment, or by painstakingly collecting "preservation metadata" in the hope that it will make future format migration possible. It turns out that *the "preservation metadata" that is needed for a format is an open source renderer for that format*. The open source community is creating these renderers for reasons that have nothing to do with preservation. Any format that has an open-source renderer is effectively immune from format obsolescence because there is no plausible scenario in which it will stop working in the current environment and, if it does, the environment in which it did work can be re-created. Further, any format that can be rendered by a binary plugin for an open source environment can be made immune simply by preserving the bits of the binary plugin.

For formats that lack an open source renderer, efforts to assure its future legibility by preserving its specification are unlikely to succeed. Resources are better devoted now to using the specifications and current access to a working renderer to remedy the lack of an open source renderer.

In addition, national libraries should collect and preserve open source repositories such as SourceForge. They are essential to their efforts to preserve other important content, such as Web crawls. There are no legal

or technical barriers to preservation; open source licenses permit all operations needed for digital preservation [17]. And who is to say that the corpus of open source is a less important cultural and historical artifact than, say, romance novels?

Of course, it must be admitted that reconstructing the entire open source software stack is not very convenient for the eventual reader, and could be expensive. Thus the practical questions about the obsolescence of the formats used by today's readers are really how convenient it will be for the eventual reader to access the content, and how much will be spent when in order to reach that level of convenience.

4.5 Virtual Machines

As we have seen (Section 3.4), even with access to both specifications and working renderers, efforts to create open source renderers for formats that are perceived to be at risk of obsolescence are likely to be unaffordable, Fortunately, another development unrelated to preservation provides a cheaper alternative. The preferred preservation strategy in Rothenberg's article was emulation, which in 1995 was a rather exotic approach. Since then virtualization, a relative of emulation, has become an essential part of mainstream IT.

We now have open source implementations of complete emulations of the Intel (e.g. [5, 13]) and other architectures. Thus, if we have the original bits for the content, a renderer, and the operating system environment in which it ran on one of these architectures we are confident that we will be able to render the content. The question is how to deliver this rendering in a user-friendly way.

Recent developments make it likely that a very convenient user experience can be delivered at low cost. For example, it is now possible to deliver an emulation of the PC architecture in Java to a reader's browser [13]. A service could easily be constructed to deliver the appropriate member of a time sequence of emulated environments, using the techniques of the Memento proposal [30]. Services could be layered on this to use the emulation of the then-current environment to render the preserved content.

4.6 Summary

The essential components of an alternate model are:

- Preserve the original bits of the content.
- Preserve the bits of the original renderer and its operating environment, to allow the preserved content to be rendered in an emulation of its original environment.
- Preserve the source code of an original open source renderer and its operating environment, both to allow the preserved content to be rendered in a reconstruction of its original environment, and to form the basis for a format converter should format migration be required.

These three components enable a robust, four-fold response to the unlikely event of format obsolescence:

- The binary of the original renderer and its operating environment could be run in an emulation of contemporary hardware.
- The original open source renderer and its environment could be rebuilt from the preserved source code and run directly on modern hardware.
- The preserved open source renderer could be rebuilt from the source code and run in a modern environment.
- The preserved open source renderer could be modified into a format converter, and run in a modern environment to generate a temporary access copy in a less obsolete format.

None of these approaches face difficulties comparable to those of the standard model.

5 Conclusions

Rothenberg's article fostered the view, which persists, that format obsolescence is a common problem that happens frequently to the majority of formats, including popular ones. Based on this diagnosis a remedy, the standard model, was developed that like all systems works well in some cases and badly in others. The last fifteen years have shown that in today's world the diagnosis is wrong; format obsolescence is a rare problem that happens infrequently to a minority of unpopular formats. Unfortunately, this means that the cases that do arise are those for which the standard model works badly, and the ones for which it works well do not arise. Because the diagnosis was, for understandable reasons, wrong the remedy is ineffective.

Fortunately, developments unrelated to digital preservation provide a robust set of techniques for ensuring the future legibility of digital documents. In practice most digital preservation systems actually use these techniques while also performing the other, doubtfully effective, tasks required by the standard model. They preserve the original bits of their content, and depend on others to preserve either or both of the source code for the relevant rendering tools, or binaries of them. If pursued alone, this approach is cheaper than the standard model for four reasons; it involves less data, requires fewer tasks, performs them later, and it leverages mainstream IT developments such as open source and virtualization. If it

were reinforced with systematic preservation of source code repositories and binaries of renderers with their operating environments, it would be more likely to be effective. Reinforcing it in this way is cheaper and more likely to succeed than trying to solve the difficulties of the standard model.

6 Acknowledgements

Thanks are due to Cliff Lynch, who initially inspired me to formulate these thoughts, Jeff Rothenberg, for constructive criticism, the commentors on the various relevant posts on my blog, and the LOCKSS engineering team. The opinions expressed, and any errors, are the author's own.

References

[1] AMENON. 0.7% HTML validity factor. http://blogs.msdn.com/oldnewthing/archive/2004/12/21/328759.aspx#332719, Dec. 2004.

[2] ANDERSON, D. Hard Drive Directions. http://www.digitalpreservation.gov/news/events/other_meetings/storage09/docs/2-4_Anderson-seagate-v3_HDtrends.pdf, Sept. 2009.

[3] ARTHUR, W. B. *Increasing Returns and Path Dependence in the Economy*. University of Michigan Press, 1994.

[4] BAKER, M., SHAH, M., ROSENTHAL, D. S. H., ROUSSOPOULOS, M., MANIATIS, P., GIULI, T., AND BUNGALE, P. A Fresh Look at the Reliability of Long-term Digital Storage. In *Proceedings of EuroSys2006* (Leuven, Belgium, Apr. 2006).

[5] BOCHS PROJECT. Bochs: The Cross-Platform IA-32 Emulator. http://bochs.sourceforge.net/, Nov. 2009.

[6] CAMILEON PROJECT. BBC Domesday. http://www.si.umich.edu/CAMILEON/domesday/domesday.html, 2009.

[7] CHRISTENSEN, C. M. *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail*. Harvard Business School Press, June 1997.

[8] GOSLING, J., ROSENTHAL, D. S. H., AND ARDEN, M. *The NeWS Book: An Introduction to the Network/Extensible Window System*. Springer, July 1989.

[9] HESLOP, H., DAVIS, S., AND WILSON, A. National Archives Green Paper: An Approach to the Preservation of Digital Records. http://www.naa.gov.au/recordkeeping/er/digital_preservation/Green_Paper.pdf, 2002.

[10] HIGHWIRE PRESS. About HighWire Press. http://highwire.stanford.edu/about/, 2009.

[11] INTERNET ARCHIVE. About Internet Archive. http://www.archive.org/about/about.php, 2009.

[12] JHOVE. JSTOR/Harvard Object Validation Environment. http://hul.harvard.edu/jhove/, Feb. 2009.

[13] JPC. JPC: The Pure Java x86 PC Emulator. http://www-jpc.physics.ox.ac.uk/, 2009.

[14] MOORE, R. L., D'AOUST, J., MCDONALD, R. H., AND MINOR, D. Disk and Tape Storage Cost Models. In *Archiving 2007* (May 2007).

[15] NATIONAL LIBRARY OF NEW ZEALAND. Metadata Extraction Tool. http://meta-extractor.sourceforge.net/, July 2007.

[16] NETCRAFT. Web Server Survey. http://news.netcraft.com/archives/web_server_survey.html, 2009.

[17] OPEN SOURCE INITIATIVE. Open Source Licenses. http://www.opensource.org/licenses, 2009.

[18] PHIL HARVEY. ExifTool. http://www.sno.phy.queensu.ca/~phil/exiftool/, Jan. 2010.

[19] POSTEL, J. RFC 793: Transmission Control Protocol. http://tools.ietf.org/html/rfc793, Sept. 1981.

[20] ROSENTHAL, D. S. H. Bit Preservation; A Solved Problem? In *iPRES2008* (Sept. 2008).

[21] ROSENTHAL, D. S. H. Format Obsolescence: Right Here Right Now? http://blog.dshr.org/2008/01/format-obsolescence-right-here-right.html, Jan. 2008.

[22] ROSENTHAL, D. S. H. Postel's Law. http://blog.dshr.org/2009/01/postels-law.html, Jan. 2009.

[23] ROSENTHAL, D. S. H., LIPKIS, T., ROBERTSON, T. S., AND MORABITO, S. Transparent format migration of preserved web content. *D-Lib Magazine 11*, 1 (Jan. 2005).

[24] ROTHENBERG, J. Ensuring the Longevity of Digital Documents. *Scientific American 272*, 1 (1995).

[25] THE FLORIDA CENTER FOR LIBRARY AUTOMATION. DAITSS Overview. http://www.fcla.edu/digitalArchive/pdfs/DAITSS.pdf, 2004.

[26] UDFR. Unified Digital Format Registry. http://www.udfr.org/, June 2009.

[27] UK NATIONAL ARCHIVES. Digital Record Object Identification. http://sourceforge.net/projects/droid/, Feb. 2010.

[28] UK NATIONAL ARCHIVES. The Technical Registry PRONOM. http://www.nationalarchives.gov.uk/PRONOM/Default.aspx, Feb. 2010.

[29] USENIX. 2009 IEEE Reynold B. Johnson Information Storage Systems Award. http://www.usenix.org/events/fast09/award.html, Feb. 2009.

[30] VAN DE SOMPEL, H., NELSON, M. L., SANDERSON, R., BALAKIREVA, L. L., AINSWORTH, S., AND SHANKAR, H. Memento: Time Travel for the Web. http://arxiv.org/abs/0911.1112, Nov. 2009.

[31] WIKIPEDIA. OpenOffice.org. http://en.wikipedia.org/wiki/Open_Office, 2009.

[32] YANO, C. How Many Journal Copies? A Preliminary Report. Presentation to ALA, June 2008.