

On the Change in Archivability of Websites Over Time

Mat Kelly, Justin F. Brunelle, Michele C. Weigle, and Michael L. Nelson

Old Dominion University, Department of Computer Science
Norfolk VA, 23529, USA
`{mkelly, jbrunelle, mweigle, mln}@cs.odu.edu`

Abstract. As web technologies evolve, web archivists work to keep up so that our digital history is preserved. Recent advances in web technologies have introduced client-side executed scripts that load data without a referential identifier or that require user interaction (e.g., content loading when the page has scrolled). These advances have made automating methods for capturing web pages more difficult. Because of the evolving schemes of publishing web pages along with the progressive capability of web preservation tools, the *archivability* of pages on the web has varied over time. In this paper we show that the archivability of a web page can be deduced from the type of page being archived, which aligns with that page's accessibility in respect to dynamic content. We show concrete examples of when these technologies were introduced by referencing mementos of pages that have persisted through a long evolution of available technologies. Identifying these reasons for the inability of these web pages to be archived in the past in respect to accessibility serves as a guide for ensuring that content that has longevity is published using good practice methods that make it available for preservation.

Keywords: Web Archiving, Digital Preservation

1 Introduction

The web has gone through a gradient yet demarcated series of phases in which interactivity has become more fluid to the end-user. Early websites were static. Adoption of JavaScript allowed the components on a web page to respond to users' actions or be manipulated in ways that made the page more usable. Ajax [9] combines multiple web technologies to give web pages the ability to perform operations asynchronously. The adoption of Ajax by web developers facilitated the fluidity of user interaction on the web. Through each phase in the progression of the web, the ability to preserve the content displayed to the user has also progressed but in a less linear trend.

A large amount of the difficulty in web archiving stems from the crawler's insufficient ability to capture content related to JavaScript. Because JavaScript is executed on the client side (i.e., within the browser after the page has loaded), it should follow that the archivability could be evaluated using a consistent replay medium. The medium used to archive (normally a web crawler tailored for archiving, e.g., Heritrix [21]) is frequently different from the medium used to replay the archive (henceforth, the *web browser*, the predominant means of

replay). The crawler creates the web archive, which is processed by a replay system (e.g., Internet Archive’s Wayback Machine [26]), which is then accessed by an end-user through the web browser medium (e.g., the user accesses Wayback’s web interface). This inconsistency between the perspective used to capture the pages versus the perspective used to view the stored content [11] introduces difficulty in evaluating the web resources’ potential to be archived (henceforth the *archivability*). Further discrepancies in the capabilities of the crawler versus the web browser make archivability difficult to measure without manual inspection.

The success of preservation of a web page is defined by how much of the originally displayed content is displayed on replay. The success of a consistent replay experience when the original experience contained a large amount of potential user interactivity (and more importantly, the loading of external resources not initially loaded) might not rely on the level of interactivity able to be re-experienced at replay. The success of this experience is dependent on whether all of the resources needed to properly display the web page on replay were captured at the time of archiving and are loaded when the archive is replayed.



Fig. 1. Google Maps is a typical example where the lack of completeness of the archive is easy to observe. Figure 1(a) shows how the web page should look. The map in the middle has interactive UI elements. An archived version of this page (Figure 1(c)) is missing UI elements (circled), and the interaction does not function. Figure 1(b) shows a recently archived (December 1, 2012) version that gives the façade of functionality when, in fact, resources on the live web are being loaded.

The nature of the execution of the archiving procedure is often to blame for not capturing resources loaded at runtime that not only manipulate the Document Object Model (DOM) but also load subsequent representations of resources. These subsequently loaded representations are often captured [21] if their location is able to be extracted from the static code by a crawler, but a problem occurs when their loading is latent or triggered by user interaction. An example of this can be seen in the Internet Archive’s capture of Google Maps¹ (Figure 1). When this archive is replayed, everything that was presented to the user (i.e., the crawler) at time of archiving is displayed. None of the trademark panning or user interface (UI) elements function in a manner similar to the live version (Figure 1(a)) of the same page. The resources, however, appear to be correctly loaded (Figure 1(c)), though the asynchronous JavaScript calls are never fired because of the broken UI elements. Versions of a page recently archived (Figure 1(b)) appear to have all resources required for interactivity, and

¹ <http://maps.google.com>

the page performs as expected, but upon inspection of the URIs, all reference the live web and not the resources at the archive.

Contrasting the completeness of the archive of an interactive website with one from a simpler website that does not contain interactive elements or the loading of external resources via JavaScript further exemplifies that this trait is to blame for archive incompleteness. For example, a page containing only HTML, images, and CSS is likely to be completely represented in the archive when preserved with an archival tool like Heritrix. In this paper we will show that the archivability of a website, given the state of the art of archiving tools, has changed over time with the increased usage of resource-loading JavaScript and the increased accessibility of websites. Further, we will examine the incapability of crawlers and archiving tools in capturing this content and what can be done to remedy their shortcomings and increase the archivability of problematic web pages.

2 Related Work

Many components contribute to the archivability of a web page ranging from reliability of the mechanism used to archive to the frequency at which the mechanism is run. Ainsworth et al. utilized web directories like DMOZ, Delicious, Bitly and search engine results to determine how much of the web is archived [2]. McCown et al., in earlier work, developed strategies for resurrecting web pages from the archives but mainly considered those with static resources (including JavaScript) [16]. McCown also touched on the sources used to recreate lost websites (of particular interest, using Internet Archive's) and the long tail effect on the unlikelihood of domain specific sites to be able to be resurrected using the larger archives as a source [17]. Mohr set the basis for the Internet Archive's Heritrix, while introducing incremental crawling into the tool's repertoire of capability [25].

As JavaScript has been the source of many problems in archiving, particularly since the web has become more dynamic, it is useful to note prior attempts relating to JavaScript and archivability. Likarish [14] developed a means of detecting JavaScript with certain facets (namely malicious code via deobfuscation) using Machine Learning techniques and Heritrix. Livshits et al. took on some of the complexities of JavaScript, including attributes only available at runtime and thus normally limited to be experience by the client [13, 15, 20, 28]. Bergman described the quantity of resources we are unable to index (the “deep Web”) [5], which has significantly increased with the advent of resources being loaded only at runtime. Ast proposed approaches to capture Ajax content using a conventional web crawler (i.e., one not used for preservation) [3].

3 Why JavaScript Makes It Difficult

A user or script normally browses the web using a user agent. In the case of a user, this is normally a web browser. Initially, web browsers were inconsistent in implementation. This inconsistency eventually was the impetus for creating web standards to remedy the guesswork developers had to do to ensure that the display was as desired. The layout engine is the component of a web browser

that is responsible for rendering HTML, the structural portion of a web page. Along with the structure, there is also a stylistic portion (implemented via CSS) and a behavioral portion (implemented in JavaScript) on the client-side.

As the layout engines of modern browsers evolved, the JavaScript rendering engine lagged behind, causing the behavioral functionality of web pages to perform inconsistently among users. This was particularly noticeable when Ajax-based websites became common. To simplify the process of obtaining the data quickly and without worry about behavior, many crawlers and scrapers do not include a JavaScript rendering engine, opting to only grab the HTML and any embedded resources and rely on the user's layout engine to render the fetched data at a later date. This is problematic in that some resources' location on the web might be built at runtime or included in the page because of JavaScript DOM manipulation. In the case of a crawler, this might be negligible, as the resource will still be hot-linked and thus included when the web page is "replayed". For crawlers intended for preservation, however, the archive must be self-contained and thus, for a complete archive, these resources must be captured and their locations rewritten to be accessible at time of replay.

Early versions of Heritrix had limited support for JavaScript. The crawler's parsing engine attempted to detect URIs in scripts, fetch the scripts and recursively repeat this process with the intention of ensuring maximum coverage of the archive creation process. Recently, Heritrix was rewritten to be built on top of PhantomJS², a headless WebKit (the layout engine used by Google Chrome, Apple Safari, etc.) with JavaScript support, which greatly increases the potential for accurate JavaScript processing and the likelihood that all resources required to replay a web page are captured by a crawler.

4 Archivability of Sites in Respect to Type

We stated in our recent work that web pages of links shared over Twitter were less archivable than those selected to be preserved by the collection-based Archive-It service [6]. Interestingly, many of the websites on Archive-It are governmental. Many of those on Twitter are commercially-based. Unlike commercial websites, governmental websites are mandated to conform to web accessibility standards for content. From this, it can be extrapolated that the more accessible a website is, the more archivable it is. Further, certain features of JavaScript (e.g., the reliance of it being enabled to show content) may make a page generally less accessible.

The markup (HTML) of a web page is rarely a hindrance to a website being captured. The difference in interpretation of markup among various users, for instance, does not make the code more or less accessible. Semantic markup, as encouraged by Section 508, WAI specifications, and other organizations that advocate accessible web development practices does affect how end-users see the content. Even if the content displayed is hidden from view, it is still likely present and thus preserved, making variance in markup replay a moot point.

In contrast to markup, behavior (usually JavaScript) can contend what content resides in the markup. If certain behaviors are not invoked, certain content may never make it to the markup, thus compromising the degree at which the content that should be archived is archived.

² <http://phantomjs.org/>

5 Experimental Setup

We used the Memento Framework [27] to query mementos with a reliable date-time association. Using a URI convention and the HTTP Accept-Datetime header, archives can be queried for the approximate time desired and the closest result will be returned.

5.1 Gauging Accessibility

While Section 508 gives suggestions on how websites should comply to be considered accessible, other organizations (namely the World Wide Web Consortium (W3C) through the Web Content Accessibility Guidelines (WCAG)) give concrete ways for developers to evaluate their creations to make them more accessible [7]. Hackett et al. go in-depth on the accessibility of archives on a pre-Ajax corpus (from 2002) and enumerate very specific features that make a page accessible in the legal sense. Much of this information is beyond the scope of this study. Other efforts were made following the document’s release on a metric to evaluate web accessibility [22], but also fail to consider an asynchronous web.

As stated in Section 4, the markup of a web page is not problematic for crawlers to capture. To understand the role that JavaScript plays in hindering comprehensive web archiving by crawlers, it is useful to examine the WCAG Principles of Accessibility [7] and remark on where issues would occur for a crawler. JavaScript, specifically, affects a page’s accessibility by hiding information from view (perceivability), only working if all components of the script are present (operability), and frequently playing a critical role in a page being useable (robustness).

```

1: function GETMEMSWITHYEARINTERVAL(mementoURIs)
2:   M ← mementoURIs[1]                                     ▷ Get First Memento
3:   lastDate ← extractDate(M)
4:   lastDateTest ← lastDate
5:   for m = 2 → length(mementoURIs) do
6:     testingDate ← extractDate(mementoURIs[m])
7:     testingDate = extractDate(mementoURIs[m])
8:     if lastDate + oneYear ≤ testingDate then
9:       if |lastDate - testingDate + oneYear| ≥
10:          |lastDateTested - lastDate + oneYear| then
11:            lastDate ← mementoURIs[m - 1]
12:          else
13:            lastDate ← mementoURIs[m]
14:          end if
15:          push(M,mementoURIs[m])
16:        else
17:          lastDateTested ← testingDate
18:        end if
19:      end for
20:      return M
21: end function

```

Fig. 2. Pseudocode to get mementos from a timegate at a temporal distance of one year per memento or as close as possible with the first memento as the pivot.

5.2 Fetching Data

The initial experiment was to test the archivability of web sites whose presence in the archive has persisted over a long period of time. These can be described as

the “stubby head” juxtaposed to McCown’s “long tail” of sites that are preserved. Alexa³ has gathered the traffic of many of these sites and ranked them in descending order. This ranking currently exists as Alexa’s Top 500 Global Sites⁴. We first attempted an approach at gathering data by querying the archives (namely, Internet Archive’s Wayback) for past Top lists⁵ but found that the location of this list was inconsistent to the present one and some of the sites in past top 10 lists remained present in the current list. We used a simple scraping scheme to grab the paginated list but that turned up pornographic sites by the third page on the 2012 list so we kept it to the top few sites to remain representative, unbiased, and to reduce the likelihood of including sites without longevity.

```

<http://api.wayback.archive.org/list/timebundle/http://cnn.com>; rel="timebundle",
<http://cnn.com>; rel="original",
<http://api.wayback.archive.org/list/timemap/link/http://cnn.com>; rel="timemap";
  type="application/link-format",
<http://api.wayback.archive.org/list/timegate/http://cnn.com>; rel="timegate",
<http://api.wayback.archive.org/memento/20000620180259/http://cnn.com>; rel="first memento";
  datetim
<http://api.wayback.archive.org/memento/20000621011731/http://cnn.com>; rel="memento";
  datetim
<http://api.wayback.archive.org/memento/20000621140928/http://cnn.com>; rel="memento";
  datetim
<http://api.wayback.archive.org/memento/20061227222050/http://www.cnn.com>;
  rel="memento";
  datetim
<http://api.wayback.archive.org/memento/20061227222134/http://www.cnn.com>;
  rel="memento";
  datetim
<http://api.wayback.archive.org/memento/20061228024612/http://www.cnn.com>;
  rel="memento";
  datetim
<http://api.wayback.archive.org/memento/20121209174923/http://www.cnn.com>;
  rel="memento";
  datetim
<http://api.wayback.archive.org/memento/20121209174944/http://www.cnn.com>;
  rel="memento";
  datetim
<http://api.wayback.archive.org/memento/20121209201112/http://www.cnn.com>;
  rel="last memento";
  datetim

```

Fig. 3. A sample abbreviated (for space) TimeMap for cnn.com

For each of these web sites, the TimeMap was acquired from Internet Archive using the URI convention “`http://api.wayback.archive.org/list/timemap/link/URI`” (where *URI* is the Fully Qualified Domain Name of the target site) to produce output like Figure 3. From this TimeMap we chose mementos with a one year spread.

The URI of each memento was passed to a PhantomJS script, and the HTTP codes of each resource as well as a snapshot were taken. A memento⁶ for `google.com` with the Memento-Datetime 20110731003335, for example, produces a line break delimited list of the subsequent HTTP codes and respective URIs dereferenced to assemble the page. Here, we noticed that subsequent requests for the resources yielded resources from the live web. We tailored the PhantomJS script to rewrite the URIs to hit the Wayback Machine instead⁷.

³ <http://www.alexa.com>

⁴ <http://www.alexa.com/topsites>

⁵ http://web.archive.org/web/2009031500000*/http://www.alexa.com/topsites

⁶ <http://api.wayback.archive.org/memento/20110731003335/http://google.com>

⁷ e.g., <http://web.archive.org/web/20110731003335/http://google.com>

This produces an identical display (part of the Wayback UI was programatically hidden) but with resource requests that access archived content. The last step was repeated but with PhantomJS sent the directive to capture the page with JavaScript off.

From the top 10 websites on Alexa for 2012, some websites had a robots.txt restriction. The number of mementos obtained by using the code in Figure 2 and applying the URI transformation produces the following quantity of mementos, ordered corresponding to Alexa’s 2012 ranking (Table 5.2).

Alexa Rank	Web Site Name	Available Mementos
1	Facebook.com	no mementos robots.txt exclusion
2	Google.com	15 mementos 1998 to 2012
3	YouTube.com	7 mementos 2006 to 2012
4	Yahoo.com	16 mementos 1997 to 2012
5	Baidu.com	no mementos robots.txt exclusion
6	Wikipedia.org	12 mementos 2001 to 2012
7	Live.com	15 mementos 1999 to 2012
8	Amazon.com	14 mementos 1999 to 2012
9	QQ.com	15 mementos 1998 to 2012
10	Twitter.com	no mementos robots.txt exclusion

Table 1. Alexa’s 2012 Top 10 websites and available mementos.

That the content of some of these websites is not preserved (namely Facebook and Twitter) by institutions has been addressed by multiple parties [1, 4, 12, 18, 19]. All of these websites may not exhibit traits of un-archivability, as previously imagined. The root domain, in this case, may not be representative of the extent at which a web site (contrasted to web page) utilized un-archivable practices. One particular site that has succumbed to the effects of unarchivability due partially to both its longevity and publishing medium is YouTube. Crook [8] went into detail about the issues in preserving multimedia resources on the web, and Prellwitz documented how quickly this multimedia degrades [23], so highlighting this website for analysis would be useful in remedying one of the many reasons that it is not sufficiently preserved.

Using the procedure described earlier in this section, we captured screen shots and HTTP requests for one memento per year of YouTube.com. While there have been efforts in attempting to capture the multimedia on this site in a reliable way (e.g., TubeKit [24]), our concern is less about executing a focused crawl and more on analyzing the results of what has been done in the past. The simpler case here of lack of archivability is observable from the homepage. In each of the cases of capturing a screen shot (Figure 4) of the memento with and without JavaScript, there is variance on the “Recently Viewed” section of the website. This part of the website is Ajax-driven, i.e., after the page has loaded, the content is fetched. A crawler could retain the JavaScript that fetches the resources and attempt to grab a copy of the resources contained within and loaded at runtime but this particular script takes a moment post-load to load and display the images that represent links to videos. This is better explained by Figure 4(a), which is representative of the memento with JavaScript enabled. The content necessary to display this section was preserved due to its reliance on runtime execution.



Fig. 4. A YouTube memento from 2006 shows a subtle distinction in display when JavaScript is enabled (Figure 4(a)) and disabled (Figure 4(b)) at the time of capture. The Ajax spinner (above each “loading” message in Figure 4(b)) is never replaced with content, which would be done were JavaScript enabled on capture. When it was enabled, the script that gathers the resources to display (blank squares in the same section of the site in Figure 4(a)) is unable to fetch the resources it needs in the context of the archive. The URIs of each of these resources (the image source) is present as an attribute of the DOM element but because it is generated post load, the crawler never fetches the resource for preservation.

Figure 4(b) shows the same memento fetched with JavaScript off. The placeholder Ajax “spinner” demonstrates that the JavaScript to overwrite the DOM elements is present in the archive and executable but the resources needed to fully build this web page do not exist in the archive.

Contrast this to five years later (2011) when a redesign of YouTube that is heavily reliant on Ajax fails. When loading this memento⁸ into Wayback via a web browser, the JavaScript errors in Figure 5(b) appear in the console. This memento (Figure 5(a)) exhibits leakage [10].

The lack of aesthetic of the 2011 YouTube memento is a result of the CSS files (first line of Figure 5(b)) returning an HTTP 302 with the final URI resulting in a 404, as evidence in the log file that accompanies the Figure 5(a) screenshot during the annual memento collection process. By examining the JavaScript log, we noted that a causal chain prevented subsequent resources from being fetched. JavaScript is fairly resilient to runtime errors and oftentimes will continue executing so long as a resource dependency is not hit⁹. The progressive increase of Ajax on YouTube over time has caused a longer chain of failures than the 2006 example. Testing this same procedure on a website that persisted from before Ajax existed until today yet chose to rely on it at one time would test whether its inclusion greatly reduced the archivability.

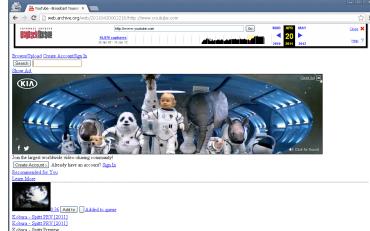
6 A Reinforcing Case

A second example where the change in archivability over time is much more dramatic can be found in the NASA website¹⁰. As a government funded agency, NASA is advised to comply with the aforementioned accessibility standards. The same procedure (Section 5) of creating a collection of annual mementos was used

⁸ <http://api.wayback.archive.org/memento/20110420002216/http://youtube.com>

⁹ This is by design of the interpreted language but appears to go against the fast-fail software philosophy.

¹⁰ <http://www.nasa.gov>



(a)

```

GET http://web.archive.org/web/20121208145112cs_/_http://s.ytimg.com/yt/cssbin
/www-core-vf1_OJqFG.css 404 (Not Found) www.youtube.com:15
GET http://web.archive.org/web/20121208145115js_/_http://s.ytimg.com/yt/jsbin/
/www-core-vf18PDCRe.js 404 (Not Found) www.youtube.com:45
Uncaught TypeError: Object #<Object> has no method 'setConfig' www.youtube.
com:56
Uncaught TypeError: Cannot read property 'home' of undefined www.youtube.com
:76
Uncaught TypeError: Cannot read property 'ajax' of undefined www.youtube.com
:86
Uncaught TypeError: Object #<Object> has no method 'setConfig' www.youtube.
com:101
Uncaught ReferenceError: _gel is not defined www.youtube.com:1784
Uncaught TypeError: Object #<Object> has no method 'setConfig' www.youtube.
com:1929
Uncaught TypeError: Cannot read property 'home' of undefined www.youtube.com
:524
GET http://web.archive.org/web/20130101024721im_/_http://i2.ytimg.com/vi/1
f7neSzDqvc/default.jpg 404 (Not Found)
  
```

(b)

Fig. 5. The 2011 capture of this YouTube.com memento demonstrates the causal chain that occurs when a resource is not captured.

to obtain screenshots (Figure 8), HTTP logs, and the HTML of the memento. Mementos ranging from 1996-2006 were available and retained, a sampling that sufficiently spanned the introduction of dynamism into the web.

The mementos from 1996 through 2002 show table-based websites devoid of JavaScript. In 2003, JavaScript was introduced into the markup (Figure 6(a)). Checkpoint 6.3¹¹ of the Web Content Accessibility Guidelines [7] mandates that pages remain usable when programmatic objects are present on the page but not necessarily supported by the user. This is to ensure all content on a page is accessible. Lack of accessibility directly correlates with unarchivability. Normally, providing an alternate means of viewing the page's content would suffice were the link to "Enter NASA" regardless of the incompatibility, but even the single relevant link on the page (with the other being a link to install Flash) is generated by a script (Figure 6(b)). From the 2004 to 2006 snapshots on, in lieu of testing for Flash, the ability to progress into the site is no longer offered but rather a message stating that JavaScript is required and a means to access instructions to enable it is the sole content supplied to the crawler. Observing the count of the resources required to construct a memento (Figure 7(a)) gives further evidence that both accessibility and archivability suffered between 2004 and 2006.

Relying solely on the number of resources fetched to determine where a site's reliance on unarchivable technologies lies is not foolproof, but it is a good guide to identify problematic pages. Were a crawler to encounter this drastic change and if the changed count was sustained, this should be noted as evidence of po-

¹¹ <http://www.w3.org/TR/WCAG10/wai-pageauth.html#tech-scripts>

```

<script language="javascript" type="text/javascript" src="flash.js"></script>
<script language="javascript" type="text/javascript">
function flashURL(id) {
    //Flash Redirect URL
    window.location.href = 'index.html';
}
</script>

```

(a)

```

var fstr = '';
if(hasFlash(6)) {
    fstr+= '<object id="screenreader.swf">...</object>';
    window.status = 'Flash 6 Detected...';
} else {
    fstr+= '...To view the enhanced version of NASA.gov, you must have
          Flash 6 installed....';
}
with(document) { open('text/html'); write(fstr); close(); }

```

(b)

Fig. 6. In 2003, `nasa.gov` introduced code (abbreviated here) into their website that checked the capability of the user's web browser and showed or hid content. The link to enter the website regardless of the user's browser capability, here, is generated with JavaScript. This would cause the content to not be displayed were the user's browser incapable or if client-side scripting were disabled in the user's browser preferences.

tential problems. On a comparable note, the same procedure was run on another government website where this deviation from web standards would be the least likely to surface, `whitehouse.gov`. A similar dip can be seen in Figure 7(b) in 2010. Examining the screen shot and log of HTTP codes, it is evident that a subset of CSS files were not preserved by the crawler. A preserved web page resembling this problem is not one necessarily related to the crawler's inability to fetch components of a page embedded in JavaScript but rather, the URI was not persistent enough to endure the time to reach Heritrix's horizon (the point at which it is preserved) once placed on the frontier (list of URIs to be preserved).

7 Conclusions

The archivability of websites has changed over time in different ways for different classes of websites. While JavaScript is partially to blame for this, it is more a problem that content is not accessible. Lack of accessibility makes content more difficult for crawlers to capture. Websites that are trend leaders, unfortunately, set a bad precedent for facilitating archivability. As this trend continues, tools are being created (Heritrix 3) that are archiving inaccessible websites. Recognizing techniques to make the archiving process easier by those that want their content preserved is a first step in guiding web development practices into producing web sites that are easier to preserve.

References

1. Twitter Donates Entire Tweet Archive to Library of Congress. <http://www.loc.gov/today/pr/2010/10-081.html>, 2010.
2. S. G. Ainsworth, A. Alsum, H. SalahEldeen, M. C. Weigle, and M. L. Nelson. How Much of the Web is Archived? In *Proceeding of the 11th Annual International ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, JCDL '11, pages 133–136, New York, NY, USA, 2011. ACM.

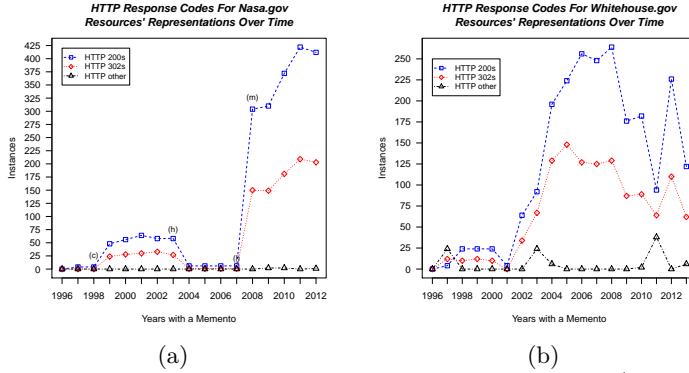


Fig. 7. The number of resources required to construct the page (Figure 7(a)) has a noticeably absent lull that corresponds to Figure 8. The preservation of the White House web page (Figure 7(b)) exhibits a different problem yet is briefly similar in that the count drastically changed. The sudden change in 2011 is the result of a set of CSS files not reaching the crawler horizon, which may have had implications on subsequent resource representations (embedded within the CSS) from being preserved.

3. P. Ast, M. Kapfenberger, and S. Hauswiesner. Crawler Approaches And Technology. [Online]. Graz University of Technology, Styria, Austria, 2008. <http://www.iicm.tugraz.at/cguel1/courses/isr/uarchive/uews2008/Ue01%20-%20Crawler-Approaches-And-Technology.pdf>.
4. J. Bass. Getting Personal: Confronting the Challenges of Archiving Personal Records in the Digital Age. Master's thesis, University of Winnipeg, 2012.
5. M. Bergman. White Paper: the Deep Web: Surfacing Hidden Value. *Journal of Electronic Publishing*, 7(1), 2001.
6. J. F. Brunelle, M. Kelly, M. C. Weigle, and M. L. Nelson. Losing the Moment: The Unarchivability of Shared Links. *Submitted for Publication*.
7. W. Chisholm, G. Vanderheiden, and I. Jacobs. Web Content Accessibility Guidelines 1.0. *Interactions*, 8(4):35–54, July 2001.
8. E. Crook. Web Archiving in a Web 2.0 World. *Electronic Library*, The, 27(5):831–836, 2009.
9. J. Garrett. Ajax: A New Approach to Web Applications. <http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications>, 2005.
10. Justin F. Brunelle. Zombies in the Archives. <http://ws-dl.blogspot.com/2012/10/2012-10-10-zombies-in-archives.html>, 2012.
11. M. Kelly. An Extensible Framework For Creating Personal Archives Of Web Resources Requiring Authentication. Master's thesis, Old Dominion University, 2012.
12. M. Kelly and M. C. Weigle. WARCreate - Create Wayback-Consumable WARC Files from Any Webpage. In *Proceedings of the 12th Annual International ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 437–438, Washington, DC, June 2012.
13. E. Kiciman and B. Livshits. AjaxScope: A Platform for Remotely Monitoring the Client-Side Behavior of Web 2.0 Applications. In *Proceedings of Symposium on Operating Systems Principles*, 2007.
14. P. Likarish and E. Jung. A Targeted Web Crawling for Building Malicious Javascript Collection. In *Proceedings of the ACM First International Workshop on Data-Intensive Software Management and Mining*, DSMM '09, pages 23–26, New York, NY, USA, 2009. ACM.
15. B. Livshits and S. Guarnieri. Gulfstream: Incremental Static Analysis for Streaming JavaScript Applications. Technical Report MSR-TR-2010-4, Microsoft, January 2010.

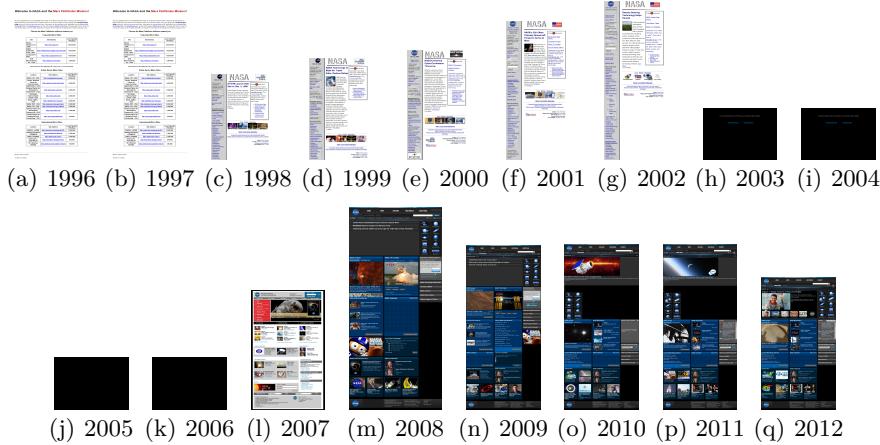


Fig. 8. NASA over time. Changes in design and thus the technologies used is easily observable between Figures 8(b) and 8(c), 8(g) and 8(h), 8(k) and 8(l), and 8(m)

16. F. McCown, N. Diawara, and M. L. Nelson. Factors Affecting Website Reconstruction from the Web Infrastructure. In *Proceedings of the 7th Annual International ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 39–48, 2007.
17. F. McCown, C. C. Marshall, and M. L. Nelson. Why Websites Are Lost (and How They’re Sometimes Found). *Communications of the ACM*, 52(11):141–145, 2009.
18. F. McCown and M. L. Nelson. What Happens When Facebook is Gone? In *Proceedings of the 9th Annual International ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 251–254, New York, NY, USA, 2009. ACM.
19. E. Meyer. Researcher Engagement with Web Archives-Challenges and Opportunities. Technical report, University of Oxford, 2010.
20. L. Meyerovich and B. Livshits. Conscript: Specifying and Enforcing Fine-Grained Security Policies for Javascript in the Browser. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 481–496. IEEE, 2010.
21. G. Mohr, M. Kimpton, M. Stack, and I. Ramitovic. Introduction to Heritrix, an Archival Quality Web Crawler. In *Proceedings of the 4th International Web Archiving Workshop (IWAW’04)*, September 2004.
22. B. Parmanto and X. Zeng. Metric for Web Accessibility Evaluation. *Journal of the American Society for Information Science and Technology*, 56(13):1394–1404, 2005.
23. M. Prellwitz and M. L. Nelson. Music Video Redundancy and Half-Life in YouTube. In *TPDL*, pages 143–150, 2011.
24. C. Shah. Tubekit: a Query-based YouTube Crawling Toolkit. In *Proceedings of the 8th Annual International ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 433–433. ACM, 2008.
25. K. Sigurdsson. Incremental Crawling with Heritrix. In *Proceedings of the 5th International Web Archiving Workshop (IWAW’05)*, 2005.
26. B. Tofel. ‘Wayback’ for Accessing Web Archives. In *Proceedings of the 7th International Web Archiving Workshop (IWAW’07)*, 2007.
27. H. Van de Sompel, M. L. Nelson, R. Sanderson, L. L. Balakireva, S. Ainsworth, and H. Shankar. Memento: Time Travel for the Web. Technical Report arXiv:0911.1112, 2009.
28. K. Vikram, A. Prateek, and B. Livshits. Ripley: Automatically Securing Web 2.0 Applications Through Replicated Execution. In *Proceedings of the 16th ACM conference on Computer and Communications Security*, pages 173–186. ACM, 2009.