Get the code here:

<http://github.com/emilyluwang/side-by-side>

In brief:

The program evaluates cache engine quality by comparing websites with their cached versions. Internet Robustness used this software to better automate the testing process. Given a website with a list of URLs that have been cached and are easily accessible (stored in a stable subdirectory), the program screenshots both the webpages at the URLS and the caches, compares the cache with the original site, and flags any sites that either failed to cache or whose caches appear dissimilar from the original site. Similarity between caches and original pages is calculated between 0 and 1.

Before running the webpage comparison tool:

**Requirements:**

* The program requires the following Python libraries, which for the most part can be installed via pip, homebrew, or easyinstall:
  + Httplib2
  + Hashlib
  + PIL (easily installed via Pillow)
  + Beautiful Soup 4
  + Scipy (better installed via Anaconda, though may cause issues with other libraries)
  + Image
  + Webkit2png must also be installed (note: not a Python library)
* My system runs Python v 2.7.6

**Assumptions:**

* Cached pages are stored at a single stable URL and can be accessed by calculating a URL’s MD5 cache value
* The HTML formatting of the page containing the URLs is fairly simple (necessary for the Beautiful Soup html parser) and all URLs are hyperlinks

**Known Limitations:**

* Cache comparison is sensitive to slight shifts within screenshots. The image comparison algorithm calculates how closely correlated the pixel colors are at each address within the image. With web page design, unlike with normal photographs, many edges are sharp. Consequently, even an offset of a few pixels can result in low correlation between some regions of the images
  + In practice, these shifts were prevent near-perfect levels of cross correlation, but lowering the baseline for a “good quality” cache still eliminated the need to check good caches. Cache quality was not necessarily bimodal, but poor quality caches did not appear until well below the recommended threshold (0.70) of cache quality
* Even if the cache doesn’t exist, the program will visit the URL of the cache anyway and screenshot it. This results in screenshots of “page not found” error messages. In these cases, the user should manuallyconfirm that the cache does not exist
* The formatting of the URL ([google.com](http://www.google.com) vs <http://www.google.com>) does not matter, as long as the URL is labeled as a link in the HTML source code (ie as long as the link is inside the <a href=…> tag)

Instructions for running the comparison tool:

* Create a folder where the screenshots will be saved and enter this filepath into the file “global\_vars.py” under the variable “img\_filepath”
  + Note that the all filepath should be entered as a relative filepaths to the location of the code
* Check that a folder called “files” exists in the same directory as the software components; this folder should contain files *results.txt* and *failed\_pages.txt*
* Set the variable “host\_url” to the url of the page containing all links to check
* Set the variable “cache\_url” to the subdirectory that contains all the caches
* Change the variable “cache\_quality” if you wish; 0.70 is generally a good threshold. Higher than 0.70 would indicate a very strict threshold for quality and would thus result in lots of manual checking and review of caches. Furthermore, the 0.70 threshold will return few or no “false positives,” ie it will still err on the side of flagging rather than not
* Run the file *check\_cache.py*
* Check the *results.txt* file for the websites with caches below the quality threshold. Check *failed\_pages.txt* for websites that the comparison tool was unable to check (presumably because the original page was not accessible for some reason).

Further information on the different files:

* *Check\_cache.py* first calls the file *screenshot\_url.py*:
  + get\_urls is called first. It visits the “host\_url” set by the user in *global\_vars.py*, examines the HTML to look for any outgoing links, and returns a list of all outgoing links
  + make\_screenshots is then called. This function visits the original site and the cached version of the sites returned by get\_urls (accessed through the MD5 hash), and screenshots both
    - Note that make\_screenshots calls the function clean\_url. We need to clean individual URLs of all backslashes because of limitations with the screenshot process (file names can’t contain backslashes)
    - File names were URLs because we needed a stable way to access files later on (during the image comparison process)
* Given the list of links to check returned by get\_urls,­*check\_cache.py* then calls functions from the file *image\_compare.py:*
  + *Image\_compare.py* first calls get on each file name (which is identical to the cleaned url of a website), converts each file from PNG to JPEG format (necessary for later comparisons), converts the JPEG image into a matrix, converts it to grayscale, and normalizes all entries in the matrix.
  + Once get has been called on all of the file names, the function compare\_images then compares the matrix of the original file with the matrix of its cache, using scipy’s correlate2d function to find the cross-correlation of the two image files. A higher correlation coefficient indicates that the matrix representations of the two files are more similar
    - Note that compare\_images has a number of error checks built in. If one of the files (either the screenshot of the cache or that of the original site) should be missing from the folder, the absence is noted in the *failed\_pages.txt* file. This sometimes happens as a result of the screenshot tool, which will occasionally be unable to access websites to screenshot them. **As a result, manual checking of the sites listed in *failed\_pages.txt* is necessary, as they have not been checked by this comparison tool.**
  + If the correlation coefficient is greater than the variable cache\_quality in *global\_vars.py*, then the result of the cross correlation will not be recorded in *results.txt*.
  + However, if the correlation coefficient is less than or equal to cache\_quality, the result is recorded in *results.txt* to be checked.

Other files in the Github Repository (<http://github.com/emilyluwang/side-by-side>):

* *generate\_urls.py* was used to scrape links off of the Balatarin site for testing. Since Balatarin’s HTML formatting nests URL’s within special h3 tags, it required a function other than get\_urls to produce a list of urls to test. Since Balatarin was a particular use case for Amber, we wanted to confirm that the cache quality for its outgoing links would be satisfactory.
* *test\_threading.py* was an unsuccessful attempt to expedite the screenshot process by using Python’s multithreading library. It did not make the process any faster. However, for those in the future who want to tinker around with multithreading, some of the framework is already in place.
* *alt\_image\_compare.py* was an unsuccessful attempt to create an alternate, faster, and more accurate method of image comparison. The alternate algorithm calculates the difference image between the screenshot of the cache and that of the original (ie the matrix consisting of the difference between the two matrix representations of the image), saves the file as a JPEG and then returns its file size. Theoretically, similar images should have more whitespace and therefore have smaller file sizes, but this was not reliable enough to use as a metric.

Potential future uses:

* The screenshot and image cross correlation functionality could be extended to quantify differences between any number of webpages based on pixel difference, though this would require altering the generate\_urls function and changing the way that image files are accessed