**Abstract**

*As more data is stored and hardware becomes more unreliable, the risk of file corruption increases. We look at the effects of bit-level file corruption on a variety of file types and examine their durability at various levels of corruption. We find a few things that seem to be true regardless of file type, such as size having an effect on durability. We also note that file types that share the same basic purpose, such as storing images, can have vastly different profiles when it comes to durability and the effects of corruption.*

**1. Introduction**

Currently, there is a huge push for “big data” and for the analysis of “big data”. This is resulting in massive amounts of data being stored and used. Inevitably, a certain percentage of data stored will be corrupted due to disk failure and other factors. While this percentage is affected by choice of file system, the fact remains that data is reliably corrupted at a rate that is above zero. As more data is stored and accessed for data mining and other “big data” research, the odds of needing data that has been corrupted increases. In addition, it has been shown that technology that improves the performance and memory size in computers can cause more hardware failures. [9]For this reason, we saw a need to study the effects that hardware corruption, emulated by random bit flipping, has on different file types in order to determine if certain file types are more resistant to corruption than others. In this paper, we use random bit flipping to emulate the corruption caused by hardware failures. We seek to examine the fault tolerance of different popular file types in order to study the resistance of various file types to hardware failures that result in bit flipping errors.

In order to accomplish this task, we developed a program that would read in a file and flip a specified number of bits ranging from 1 to 1,000. The specific bits that were flipped were randomly selected using a random number generator. After processing a file through our “bit flipper”, we ranked it by usability. We performed many repetitions of this and various file types, and so were able to draw conclusions on which file types and sizes were more resistant to hardware failure resulting in file corruption. Our full methodology is documented in section 4.

**Contributions:** We make the following contributions:

We created a program to emulate the effects of hardware failure on a variety of file types

We establish a rating scale to score the usability of a corrupted file

We detail the resistance of different file types and sizes to corruption due to hardware failure

**2. Background**

It has been proposed that future programming models will need to take into account increasingly unreliable hardware and that rather than creating hardware-specific methods of dealing with this unreliability, portable methods need to be developed. [3]

In addition, failure recovery can sometimes introduce unexpected problems into a system, so it is sometimes better to react conservatively. Sometimes it may be better to ignore a problem if it does not affect the functionality of the system or the usability of the file. [2]

For these reason, we felt it important to be aware of how various file types fared when corrupted, as file types are standardized and will suffer the same problems regardless of the method of corruption. If a system was built only to monitor the file types least resistant to corruption, it would be less CPU intensive and less intrusive than a system that monitors every file in the system. Also, companies that deal with a large amount of files of a specific type would be aware of how dangerous small hardware failures can be.

**3. Prior Studies**

Most prior studies on corruption have focused not on the effects of it, but on detecting and repairing corruption.

Although there are many methods available for the detection of faults, error correction coding is fairly common. This may be because using error correction codes can not only detect errors, but corrects them. There are many types of error correcting codes used in a variety of applications in a variety of fields. The study of these has become a rich field over the course of time. [5]

There have been studies on methods of software recovery of hardware problems, such as Relax and SWIFT. Both of these were created because of the rising unreliability of hardware and the advantages a software-based approach has over hardware-based recovery. [1, 8] Additionally, there has been research on tools that can transform an application into a different application that fulfills all of the functions of the first action and can additionally detect bit-flip errors in some of the more sensitive areas of a computer (registers, memory, etc.). The C2C Translator is a prototypical tool created to do just that with source code written in the C language. [6]

This paper does not focus on detecting or repairing corruption, but simply on what corruption looks like in various file types and how usable they are after they are corrupted. While there has been research on the effects of errors in other pieces of the system, such as the register file and the cache, we do not believe file corruption has been studied at the file type level. [4, 7]

**4. Methodology**

We chose to examine the following file types:

.txt

.doc

.docx

.pdf

.png

.bmp

.jpg

.exe

For each file type, we tested in multiple files ranging in length and/or complexity. For each individual file, we flipped 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 50, 100, 500, and 1000 random bits. For each number of bits flipped, we produced ten corrupted files. Correspondingly, each file that went into the bit flipper resulted in the output of 170 different corrupted files: 10 files with 1 bit flipped, 10 files with 2 bits flipped, 10 files with 3 bits flipped, and so on.

In this paper, a test case refers to the combination of a file and a number of bits flipped. We chose to run each test case ten times. Clearly a single file cannot be representative of an entire test case. Similarly, the more files we examine for each test case the more accurate our usability score will become. We chose to run each test case ten time so that we could get an accurate representation of what the average test case would look like, while still keeping the scale small enough that we would be able to visually inspect each file.

Each file that the bit flipper output was visually examined and scored from 0 to 5, with 0 being unopenable and 5 being perfect. Further details on scoring can be found in section 4.2. For each test case, an average usability score was calculated and this average score is the value referred to as the **usability score** for the duration of this paper. As the usability score is a mean and a mean is not always the best measure of central tendency, there are instances where we will delve further into the data and judge it in different ways.

In addition to calculating the usability score for each test case, we kept track of the percent of files that opened for each test case. As we ran each test case ten times, this number is always a nice round percentage.

**4.1 Flipping Bits**

<code snippet and discussion>

<adam write here!>

**4.2 Scoring Files**

Every file was ranked from 0 to 5 as follows:

0 – The file does not open at all. A window may open, but no part of the file loads.

1- The file opens but is corrupted beyond usability.

2- The file opens and is obviously corrupted, but recovery of some data is possible.

3- The file opens and nearly all data is intact, but some corruption is easily detected.

4- The file opens and most data is intact, a diff is required to find any change in the file.

5- The file opens and even a diff cannot detect a change.

While the above scoring system guided all efforts, further descriptions will be given for each file type examined and how the test files of the specific file type were judged to fall into a given score.

There is more granularity in this score system for files that function well than for those that function poorly. We briefly reviewed the test files before we developed the rating scale and we noticed a need for more granularity among files with a high level of usability.

An important aspect of this scoring system is that we focused not on some percentage of characters changed or a purely quantitative measure, but on the actual usability of the document. This is something of a rarity in computer science, but we felt it was more important to understand how useful the corrupted document still is than to generate statistics on how much of the document changes.

As there were several thousand files to be examined, we divided the scoring among the three members of the research group. In order to ensure consistency of scoring within a single file type, a single person scored all files for a given file type. All document formats were handled by a single person, all executables by a different person, and the image files were split among two people, but a single person handled all of a given file extension.

**4.3 Methods Limitations**

We did not examine cross-scorer validity <This isn’t the name for it. Having a brain fart… >

The subjective nature of our scoring system leaves blah blah blah

There is not a fine level of granularity in our scoring system. Two different test cases scored as two could still look vastly different as it covers all files that function and allow for the recovery of some data through files that are clearly corrupted.

**5. Results – Document File Types**

**5.1 .txt Files**

.txt files were unique in that all corrupted files could still be opened. This is likely the result of a small header that that rarely got hit by a random bit flip, and which didn’t affect the file’s ability to be opened if it did hit the header. They were also unique in that every single file showed some evidence of corruption. While the evidence was frequently very small and could only be detected by using a diff, every file had some change in the viewable text from the original file.

Text files were also unique in that they were the only file type where we saw a complete change in character set. This occurred several times in the small files. When this occurred, the file was scored as a 1.

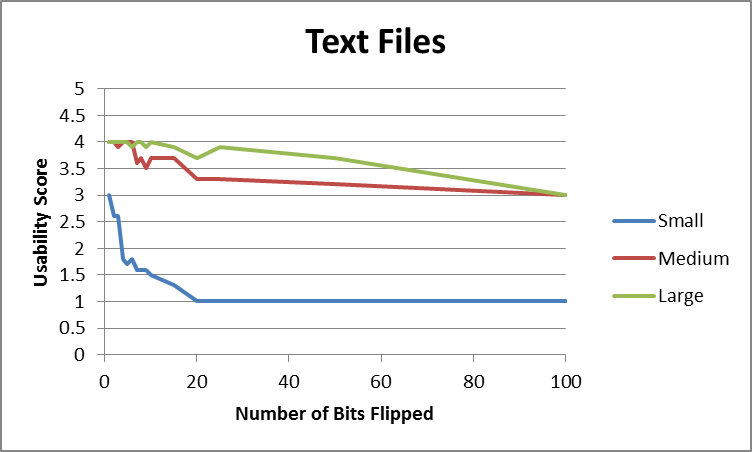


Figure 1 – The deterioration of text files. Note the ordering – small files deteriorating faster than medium files, and medium files faster than large files. Note also that at no point does any file score a zero or a five.

As demonstrated in figure 1, it was readily apparent in text files that the smaller the file was, the more rapid the deterioration of usability. This observation is logical, as given that the number of flipped bits holds consistent, the smaller the file size, the higher the density of the flipped bits. As the small file was only five words, it deteriorated very quickly. The medium file was five pages, which is quite a jump from 5 words, and while it did deteriorate slightly more quickly than the large file, it was still a very slow deterioration and not nearly as dramatic as the deterioration of the small file.

When viewing the text files, all medium and large files fell into the range of 2-4, with the vast majority of files being scored as threes or fours. The small files deteriorated down to a usability score of one, but the medium and large files both maintained very high functionality, never even falling below a usability rating of three. The medium and large files were still completely readable, with only misspellings and non-alphanumeric numbers serving as evidence of their corruption. Up until about 50 bits flipped, a diff was frequently required to find any difference between the corrupted and non-corrupted files in the small and large test files.

**5.2 .doc files**

The performance of the .doc files surprised the research group. We expected the .docs to fail quickly, but were surprised by the number of the corrupted files that could be opened. As opposed to the text files where all files could be opened and none of them were perfect, not all of the .doc files could be opened and there were multiple examples of perfect files. In fact, many of the small .doc files with only a few bits flipped received a usability score of 5, meaning that we could not detect, even with a diff, any change in the document. This would indicate that the bit flip occurred in some part of the header that was not accessed.

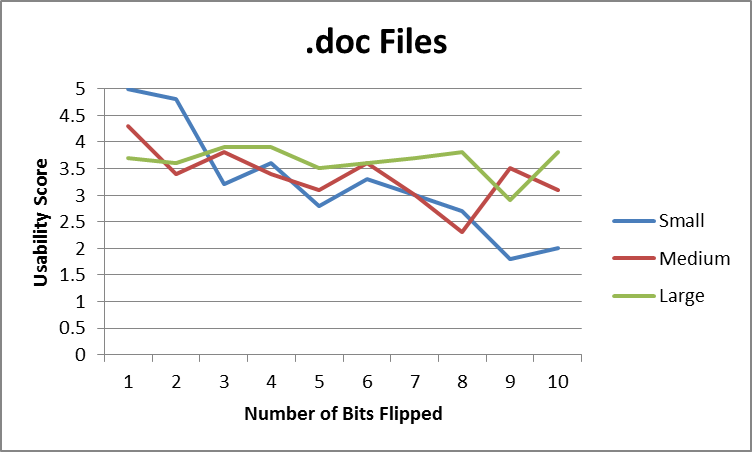


Figure 2 – Small, Medium, and Large .doc files are close together in functionality.

Looking only at the average usability scores of the files with 1-10 bits flipped, shown in figure 2, all three file sizes rank quite closely to each other. The small file starts with a higher usability, but by the point three bits are flipped, the files fall into the expected order – with the large files being the most usable and the small files the least. As shown in figure 3, the files mostly stay in that order as more and more bits are flipped, with the ordering becoming more consistent as more bits are flipped.

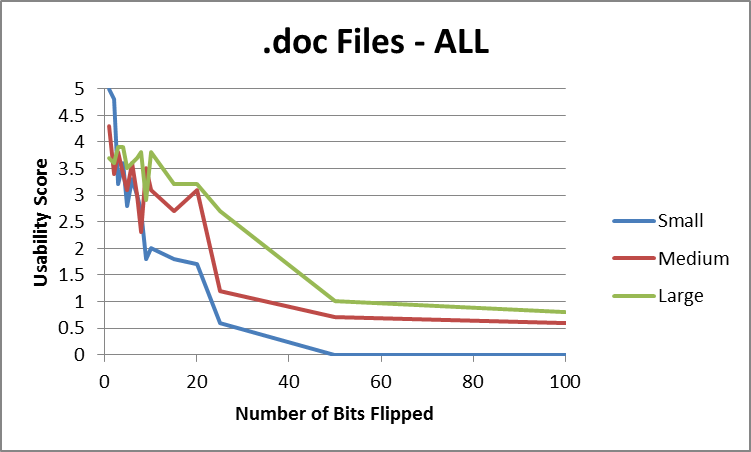


Figure 3 – Small, Medium, and Large .doc files spread apart as more bits are flipped.

Looking only at the average scores fails to show some interesting trends in the performance of the .doc files. The small .doc test files tended to score either a score of five or of zero, with only a few files falling in between. The small .doc files were also the only .doc files where test files scored fives. Additionally, it wasn’t just one or two test files that scored fives, but all ten files with one bit flipped, nine with two bits flipped, and six with three bits flipped. This is quite a difference from the medium and large files, where not a single file was perfect. I suspect that this is because due to the lack of content in the small files, the bit blip was most likely to fall within the header where it would either make the file completely unreadable, or wouldn’t affect the file at all.

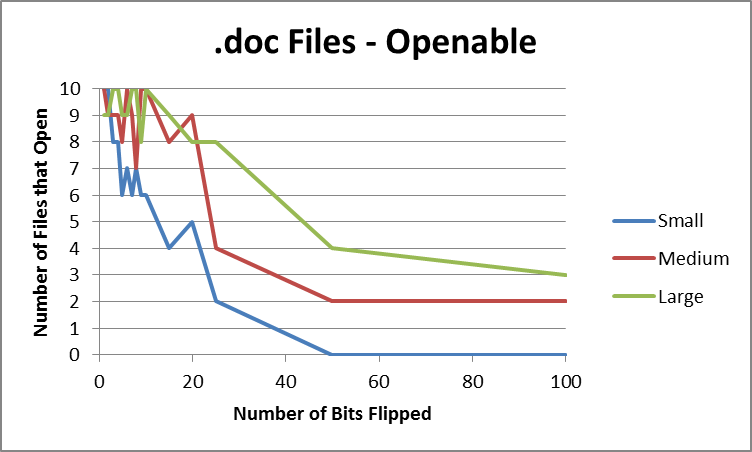


Figure 4 – Graph showing the number of.doc files that opened across file size and number of bits flipped.

Figure 4 shows the rate at which files failed to open for the small, medium, and large .doc files. You can see that the small files fell of more quickly than the medium and large files did. In conclusion, small files were more likely to function perfectly and more likely to fail to open than both medium and large files.

While the medium and large .doc files did not have any examples of perfect files, there were plenty of examples of high functioning files. If the file opened, it was almost always scored either a 3 or a 4, meaning it was easily readable and any missing or changed data could be inferred. As mentioned when discussing the functioning of the small files, the medium and large files were both more likely to open and significantly less likely to function perfectly than small files.

**5.3 .docx files**

The .docx files were the easiest to score, as very few of them opened. Any docx file that did open was completely perfect and scored a 5.

Among the small files, only two files opened, both of them having only one bit flipped. This resulted in an average usability score of 1 for the small.docx files with 1 bit flipped. This is again an instance in which the mean is a bad measure of central tendency as it overestimates 80% of the data and strongly underestimates 20% of the data. Perhaps more revealing is that the median and the mode are both zero and the standard deviation is 2. The usability score for all tests beyond one bit flipped is zero.

For both the medium and large files, only a single file opened. In both instances, the single file was one of the ten with only bit flipped. This resulted in a usability score of .5 for a single bit flipped, and of 0 for all other cases.

**5.4 .pdf files**

PDF files were the most interesting to score as they were the only file type where flat out strange things happened. Fonts changed from page to page, spacing and formatting were off, and entire blocks of text would disappear.

Anytime a PDF appeared to be perfect and I ran it through a diff, it turned out to be perfect. When a PDF corrupted, it wasn’t subtle about it. Either formatting obviously changed or a block of text disappeared, leaving a big blank space. This means that there wasn’t a single PDF with a score of 4.

Another factor that was unique to PDFs was the number of errors that they threw. Scoring the PDFs took significantly more time, simply because for a single document, I may have had to click through 15-20 errors in order to see the whole file. It seemed that it notified me that it detected corruption not once for the entire file, as Microsoft Word did, but each time that it detected some corruption as I scrolled through the file.

PDFs were the only file type that would open a window and present a blank document, but fail to present any text from that document. When this occurred, the file was marked as if it did not open at all as there was no text of any sort rendered.

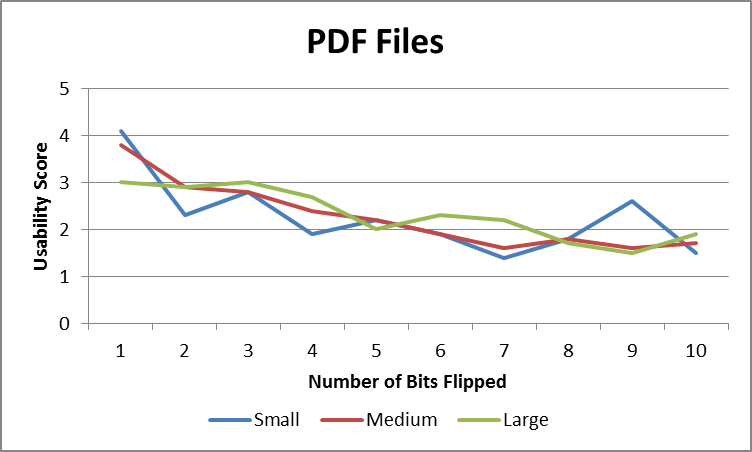


Figure 5 – Deterioration of PDF files with 1-10 bits flipped. Note how close the small, medium, and large files are in their rates of deterioration.

By usability score, PDF files deteriorated more consistently than others and the small, medium, and large files deteriorated at roughly the same rate. Figure 5 shows the rate of deterioration of PDF files with 1-10 bits flipped. This finding is in contrast to the other file types where small deteriorated significantly more quickly than medium and medium deteriorated more quickly than large. While the usability scores fail to show small deteriorating significantly more quickly, the open rates do. As show in figure 6, even from a single bit flipped, the small PDFs are less likely to open than the medium or large ones.

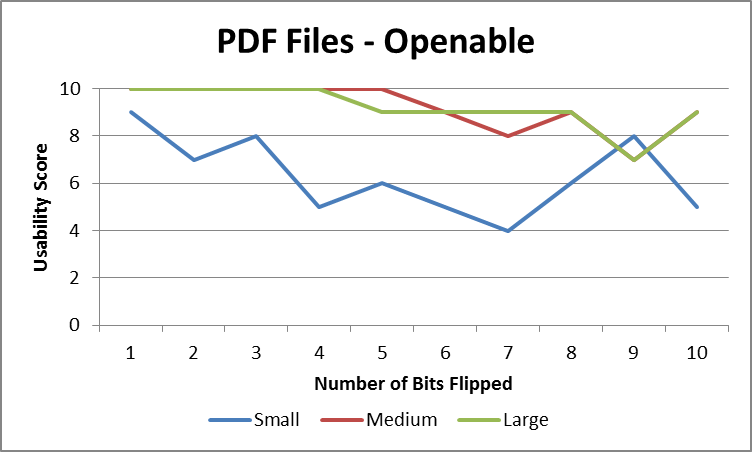


Figure 6 – Note the rapid deterioration of the open rates of small files compared to medium and large files.

The last observation unique to PDF files is shown in figure 7. Figure 7 shows that in the long term, the small files function better than the medium files, which function better than the large files. This is the precise opposite of all the ordering of all other file types, in which the small files deteriorated most rapidly.

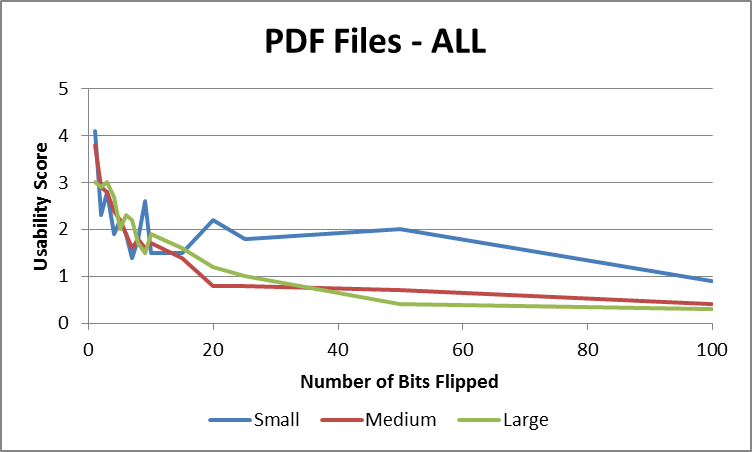


Figure 7 – Note the ordering of usability of the small, medium, and large files. This is the opposite of the ordering for all other document file types.

**5.6 Comparison of Document File Types**

Regardless of the size of the file, the open rates of files follow the same trends. Test files always open, docx files almost never open, and doc and pdf files fall somewhere in the middle. PDF files have a higher open rate than doc files by quite a bit for the small files. The same holds for medium files, though the margin is smaller. Doc files narrowly overtake pdf files and are more openable with large file sizes.

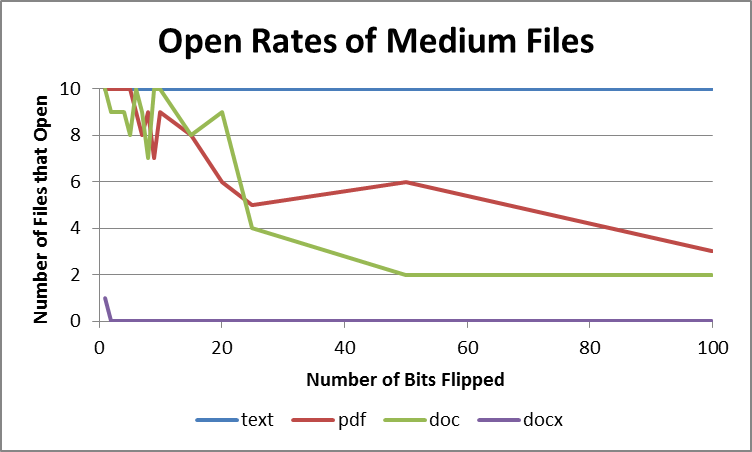


Figure 8 – The number of medium text, pdf, doc, and docx files that open at 1-100 bits flipped. Compare to figure 9, displaying the open rates of large files.

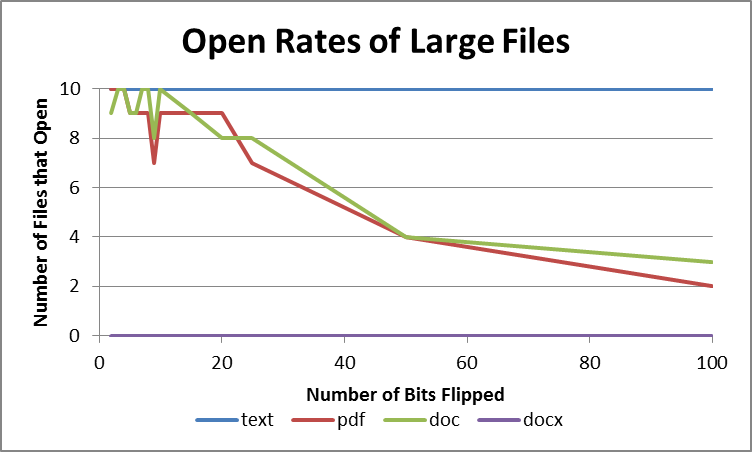


Figure 9 - The number of large text, pdf, doc, and docx files that open at 1-100 bits flipped. Compare to figure 8, displaying the open rates of medium files.

While the open rates for small, medium, and large files followed somewhat similar trends, the usability graph for small files looks completely different than the usability graphs for medium and large files.

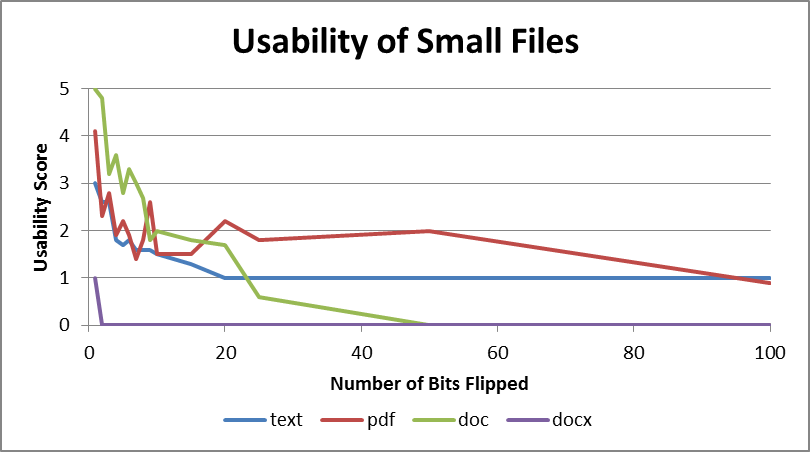


Figure 10 – Usability of small files with 1-100 bits flipped. Compare to the graph in figure 11 plotting the usability of large files with 1-100 bits flipped.

The most unique feature of the graph of the usability of small files, shown in figure 10, is the precipitous drop of usability of .doc files. This is unique to the small files. The .doc files do not start at such a high point, nor fall so quickly to such a low point in the medium and large files. I hypothesize that the sharp drop is due to the fact that there is such a small amount of actual text to store, that the likelihood of the flipped bits being located in the header are increased. When a bit flip hits the header, it is likely to result in a file that cannot be opened. For more detail, see the above section dedicated to .doc files.

Another unique feature of the small files is the poor usability score of the text files. In both medium and large files, the text files maintain the highest usability score regardless of the number of bits flipped. The reasons small text files deteriorated so quickly are covered in more depth in the above section dedicated to text files.

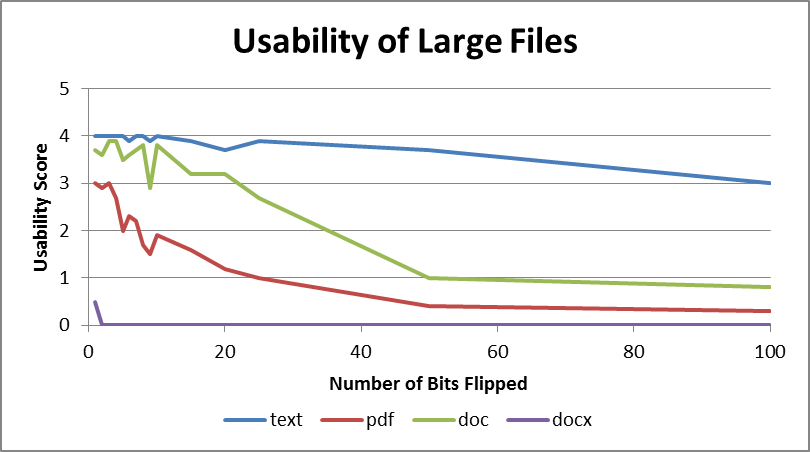


Figure 11 – Usability of large files. Note that the graph of medium files is similar, will less difference between pdf and .doc files.

**.BMP files**

Bitmaps generally opened and most changes were minor. It appears that the smaller file sizes of bitmap did better earlier, but in the higher levels of bit flipping the larger sized files did better. Most of the issues with bitmap were undetectable to the naked eye. However, when they were run through ImageMagick’s image compare tool, which highlighted any pixels that were different, there were definite changes to the image. One interesting aspect of bitmap files for which I cannot find an explanation was that all of the pixel differences occurred in the bottom 25% of the picture. Our initial response was to think that there was a problem with the bit-flipping program and it was not flipping random bits. However, upon comparing a bit-flipped version with the original in a hex editor, it was discovered that the differences occurred throughout the file. There is nothing in the specifications for bitmap that describes why pixel changes occurred in the lower portion of the picture, but this was the case throughout the bitmap files. If the file opened, the file was the right size, and the file did not have a complete color shift, the changes were all in this lower portion.

**.GIF files**

As expected, the GIF files did not degrade as gracefully as the bitmap files. While there were some errors that were undetectable to the naked eye, particularly in the solid colored files, many were obvious at first glance. Some of the errors in files that were able to be opened included complete color shifts, large portions of “static,” distortions that made it appear one was looking at part of the picture underwater, and entire portions of the picture being in the wrong place. As before, large files fared better than smaller files. With the quad-color file in particular, if the file opened it generally looked extremely distorted, even with only one bit flipped. Meanwhile, the full-color photos tended to be somewhat recognizable, although distorted, even with as many as 20 bits flipped.

**.JPG files**

Surprisingly, JPEG files did better than GIF files overall. Much of this was likely due to the types of errors that occurred. In JPEGs, the errors tended to come in three types if the file was able to be opened. The first type was a simple color shift, where the file appeared to be mostly normal, with a colored filter placed over it. The next type, which often accompanied the first, was a simple shifting over of portions of the picture. The last type was a solid colored area covering a portion of the picture. The four-colored quadrants again did the worst out of all of the files.

**6. Implications**

**6.1 Future Research / Unanswered Questions**

**7. Conclusions**

**References**

[1] DE KRUIJF, M., NOMURA, S. AND SANKARALINGAM, K. 2010. Relax: An architectural framework for software recovery of hardware faults. In *ACM SIGARCH Computer Architecture News,* Anonymous ACM, , 497-508.

[2] FUJITA, H., SCHREIBER, R. AND CHIEN, A.A. 2013. It’s Time for New Programming Models for Unreliable Hardware. *Provocative Ideas Session, ASPLOS* .

[3] GUO, Z., MCDIRMID, S., YANG, M., ZHUANG, L., ZHANG, P., LUO, Y., BERGAN, T., BODIK, P., MUSUVATHI, M. AND ZHANG, Z. 2013. Failure recovery: When the cure is worse than the disease. In *Workshop on Hot Topics in Operating Systems, HotOS XIV. USENIX Association,* Anonymous .

[4] KIM, S. AND SOMANI, A.K. 1999. Area efficient architectures for information integrity in cache memories. *ACM SIGARCH Computer Architecture News* 27, 246-255. .

[5] MOON, T.K. 2005. Error correction coding. *Mathematical Methods and Algorithms.Jhon Wiley and Son* .

[6] NICOLESCU, B. AND VELAZCO, R. 2003. Detecting soft errors by a purely software approach: method, tools and experimental results. In *Design, Automation and Test in Europe Conference and Exhibition, 2003,* Anonymous IEEE, , 57-62 suppl.

[7] REBAUDENGO, M., SONZA REORDA, M. AND VIOLANTE, M. 2003. An accurate analysis of the effects of soft errors in the instruction and data caches of a pipelined microprocessor. In *Proceedings of the conference on Design, Automation and Test in Europe-Volume 1,* Anonymous IEEE Computer Society, , 10602.

[8] REIS, G.A., CHANG, J., VACHHARAJANI, N., RANGAN, R. AND AUGUST, D.I. 2005. SWIFT: Software implemented fault tolerance. In *Proceedings of the international symposium on Code generation and optimization,* Anonymous IEEE Computer Society, , 243-254.

[9] SHIVAKUMAR, P., KISTLER, M., KECKLER, S.W., BURGER, D. AND ALVISI, L. 2002. Modeling the effect of technology trends on the soft error rate of combinational logic. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on,* Anonymous IEEE, , 389-398.