Journal of Computer Sciences and Applications, 2018, Vol. 6, No. 1, 43-47
Available online at http://pubs.sciepub.com/jcsa/6/1/6
©Science and Education Publishing
DOI:10.12691/jcsa-6-1-6

DotCode Damage Testing

Kevin Berisso[*]

Department of Engineering Technology, University of Memphis, Memphis USA
*Corresponding author: kberisso@memphis.edu

Abstract The DotCode bar code symbology is relatively new, having been developed by Dr. Andrew Longacre in 2007. As a result of standards organization work being done on the symbology, it was determined that valid encoding patterns could result in symbols that could not be decoded by the current reference decode algorithm. A solution for resolving the issue involved the introduction of intentional errors that rely on the correct functioning of the Reed-Solomon Error Correction (RSEC) to resolve the original message. However, the actual impacts of such a decision were publicly unknown. This paper has undertaken the task of validating the impacts of intentionally adding additional data bits to the bar code, resulting in the generation of errors in what should otherwise be perfect symbols.

Keywords: *DotCode, bar code, 2D bar code, reed-solomon error correction*

Cite This Article: Kevin Berisso, "DotCode Damage Testing." *Journal of Computer Sciences and Applications*, vol. 6, no. 1 (2018): 43-47. doi: 10.12691/jcsa-6-1-6.

1. Introduction

The DotCode bar code symbology was developed to address problems that can occur within high speed printing applications in the supply chain [1]. Comprised of a series of dots on a diagonal grid (see Figure 1), the symbology is inherently robust to changes in line speed, clogged ink jets and other printing problems.

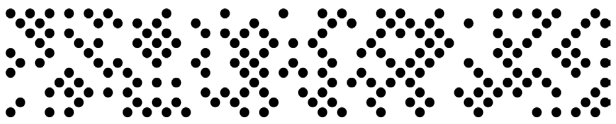
Figure 1. Example of the DotCode symbology. The "Barcode Scanners" app by Cognex can be used to scan DotCode using either Apple or Android devices

During testing by Mr. Terry Burton of his Postscript encoder [2], it was found that some instances of the processed data resulted in encoding patterns that generated symbols with entire outer rows or columns that were blank. Due to the nature of how DotCode operates, a missing row and or column would result in an otherwise properly encoded and printed symbol but for which the reference decode algorithm would fail to decode, as seen in Figure 2.

To combat this issue, Mr. Richard Skokowski suggested that the encoders always print some of the six dots that make up the four corner positions of the symbol as a way to avoid unprinted edges [3]. It was suggested that even though this would intentionally impart Reed-Solomon Error Correction (RSEC) errors into the symbol prior to printing, the benefits would significantly outweigh the danger of generating a symbol that had an entire outer row or column of dots missing automatically resulting in a symbol that could not be scanned without a more sophisticated decode algorithm than the current standard that is already deployed.

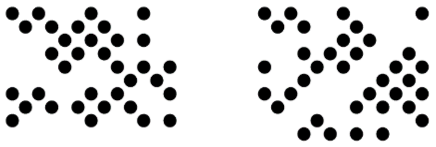
Figure 2. Symbols with the same data ("874130"), but using different masks resulting in a missing bottom row (left) which prevents the symbol from being decoded whereas the right symbol can be decoded

This paper reports on the results of testing that occurred at the University of Memphis. The testing attempted to determine if the assumption that the intentionally introduced RSEC errors would have a negligible impact on the performance of bar codes scanned at simulated production-level speeds.

2. Discussion

The DotCode symbology is a part of the matrix symbology family of bar codes. This family of symbologies (types of bar codes) is most easily identified by the clear mapping of modules in some sort of regular grid or matrix pattern. Examples of commonly seen matrix symbologies include QR Codes, Data Matrix, Maxicode and Aztec Code (see Figure 3).

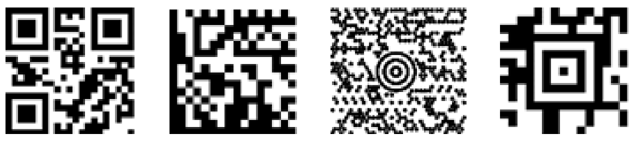
Figure 3. Samples of some common matrix symbologies, QR Code (left), Data Matrix (middle left), Maxicode (middle right) and Aztec Code (right)

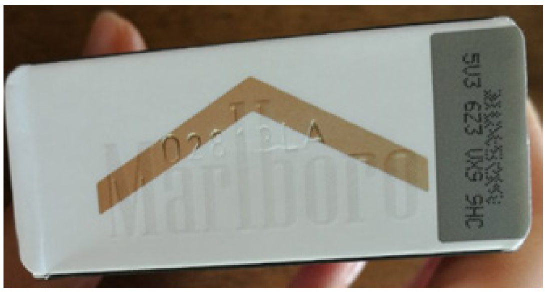

Figure 4. (Color online) DotCode on a pack of cigarettes purchased in Greece. Picture taken by Alexandra Valtzidou

Experiencing adoption at some European cigarette manufacturing facilities, DotCode is allowing companies and government entities to apply unique, serialized machine-readable codes onto products that are run on high speed manufacturing and labeling lines (see Figure 4). Due to the importance of easily decodable bar coded data on products [4-9], those industries that print bar codes on-demand at the time of manufacture or packaging need solutions that are robust enough to ensure quality printing during the often bumpy and turbulent manufacturing process. DotCode was specifically developed with a separated module design to help absorb any inadvertent movement of the printing target (e.g. label, box, etc.) during the application of the bar code [1].

Specifically characterized as a 5-of-9 symbology, where five out of every nine data dots are black, DotCode is capable of encoding 112 unique code words in one of three code sets in a simple on/off pattern. The pattern is encoded within a matrix that is characterized by having a diagonal grid that is reminiscent of a checker or chess board and for which the number of rows and columns always add up to an odd sum. The symbology employs a RSEC methodology that consumes approximately 33% of the overall symbol size and is capable of encoding various character sets via the optional extended channel interpretation mechanism [1].

2.1. Encodation Method

DotCode is encoded by converting the data into a series of 5-of-9 dot patterns per the rules of the AIM International Symbology Standard Information technology - Automatic identification and data capture techniques - Bar code symbology specification - DotCode [1]. Data compactness is achieved via the selective use of shifts and latches within the three code sets (A, B and C) and logic rules that are provided to help ensure encoding efficiency. Once the data has been encoded, the correct number of RSEC codewords is added to the data stream, per Equation (1) from the standard.

$$\mathrm{NC} = 3 + (\mathrm{ND\ div\ } 2) \tag{1}$$

where,
NC = Number of RSEC code words
ND = Number of data code words

The resulting codewords are converted to their binary coded decimal nine-dot patterns, the addition of padding codewords is determined and two dots are reserved for the mask indicator which is discussed in the next section. The minimum final dot count is determined by Equation (2)

$$\text{Min Dot Count} = 9 \text{ x} \left(\mathrm{ND} + 3 + (\mathrm{ND\ div\ } 2)\right) + 2 \text{ data dots} \tag{2}$$

where,
ND = Number of data code words

2.2. Masks and Scores

Once the dot stream has been generated, the resulting pattern is processed through the normative scoring routine provided in the specification and compared against a nominal threshold that is calculated as

$$\mathit{Threshhold} = \frac{\mathit{rows\ x\ columns}}{2}. \tag{3}$$

The goal of the scoring is to prevent instances where there are large gaps in the dot pattern that could potentially impact the ability of the symbol to be decoded. Per the specification, the goal is to have the symbol's score exceed the calculated threshold. As seen in Figure 5, by applying one of four masking methodologies to the symbol, missing columns in the symbols can be minimalized if not completely eliminated.

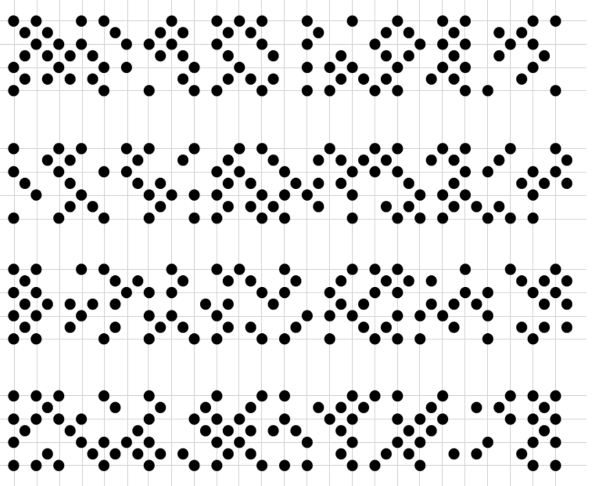

Figure 5. Symbols with the same data ("0242FZ5CGW0D"), encoded in a 7 row by 50 column pattern using masks 0-3 (top to bottom) showing how the masks can eliminate blank columns (masks 0 and 4). The scores for each mask are -106, 137, 325 and -15 respectively and the threshold is 175

2.3. Placing the Dots

The DotCode specification indicates that a properly printed symbol must be configured such that the sum of the rows and columns result in an odd number. This allows for an unambiguous determination as to whether the symbol is encoded starting at the upper left corner or the lower left corner [1]. If the number of rows are even then data is encoded in the columns, moving from left to right and top to bottom. If the number of rows are odd then the data is encoded moving from left to right starting at the bottom row and progressing to the top row. Figure 6 shows the two encoding patterns.

2.4. Unlit Edge Issue

Due to the encodation pattern shown above in Figure 6, it is possible to generate DotCode symbols that have missing edges; where all dots are in the "off" state. In instances where the missing edge is the "second edge" (dots 4, 8 and 12 in Figure 6), the current decode logic may be able to compensate for this. However, when the "first edge" (dots 1 and 2 in Figure 6) where the mask is identified or when multiple edges are missing, the bar code scanner's ability to determine how to decode the information becomes sufficiently compromised, resulting in an inability to decode the information. Figure 7, shows such an example.

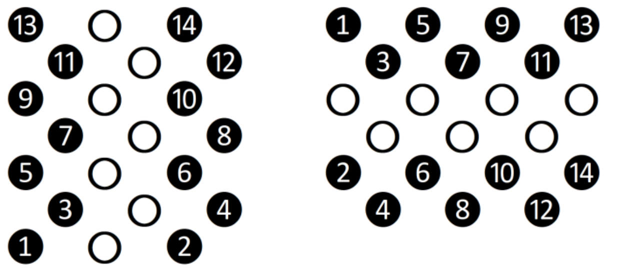

Figure 6. Two symbols showing the sequence (the numbered circles) and direction in which the dots are encoded. The left pattern is 7 rows x 6 columns and the right image is 6 rows by 7 columns

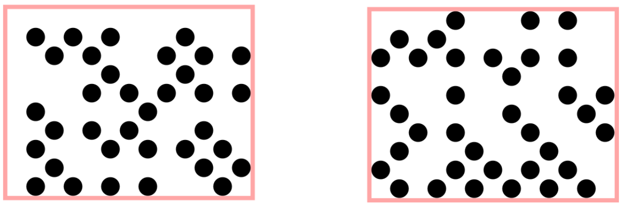

Figure 7. (Color online) Symbols with the same data ("038451"), encoded in a 10 row by 13 column pattern using masks 0 (left) and 3 (right) showing how a symbol could inadvertently be generated that is missing multiple edges. The light red boarders indicate the edges of the grid

It is due to this potential that it was proposed that all four corners always be asserted (the dots are always black) whenever any of the edges were unlit. The result would be that, in Figure 6, dots 1, 2, 4, 12, 14 and 13 would always be "on" (having a binary value of one). In these instances this would result in additional dots in the stream, resulting in the triggering of the RSEC logic during the decode process; potentially resulting in an increase to the overall decode time.

3. Methodology

To determine the impacts of various levels of intentional errors on symbols though the illumination of all four corners, a high-speed conveyor simulator was developed and scanners were positioned above the path as shown in Figure 8. The simulator has 30 stations on which 1" x 4" labels could be attached. The system is capable of speeds of up to 300 feet per minute for a resulting maximum "box" rate of 30 boxes per second. The bar code scanners were configured to only scan DotCode symbols and had their window of interest (the portion of the image that was processed) limited such that only the symbols, and not the entire label, were processed as a way help improve the processing time required. The reason for this was that at 30 scans per second, the scanner only had 33 milliseconds in which to acquire, decode and fix scanned bar codes. It should be noted at this time that the scanners used were configured to the best of the author's abilities but due to budgetary limitations, the proper external strobe lighting the application called for was not available. As a result, the performance of the two scanners used may not be truly representative of the equipment's performance potential.

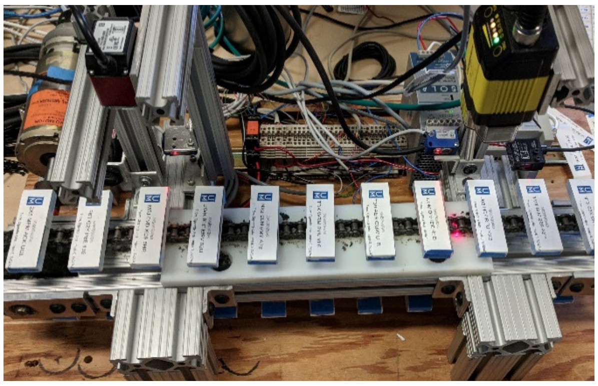

Figure 8. (Color online) Testing rig. Visit vimeo.com/229003022 for video footage of the system

The scanners were configured to transmit the bar code data and the scanner's internally reported decode time, a diagnostic tool that helps the hardware integrator to determine if the scanner is optimally aligned. Data collection was accomplished via the scanner's serial ports and physical serial ports (not USB to serial adapters) on the data collection computers. The free software package RealTerm (sourceforge.net/projects/realterm/) was used to log the data. The system originally used custom data logging software. However due to inefficiencies in the program, data was being corrupted or lost, resulting in the decision to switch to RealTerm.

Twenty-nine unique bar code symbols were generated in Seagull Scientific's Bartender Designer 2016 R3 software. For instances where intentional damage was being introduced, the symbols were exported out of Bartender and imported into Adobe Photoshop where the necessary additions and erasures were generated (see Figure 9). The images where then printed on a color laser printer and were attached to the test rig. A symbol encoding the word "UOMDIVIDER0F" was placed in the 30th test position to provide the researchers with an index point in the data stream. Efforts were made to ensure the labels were placed in the same place each time, but due to the manual nature of the process, the window of interest for each scanner was also updated to help ensure as consistent a decode time as was possible.

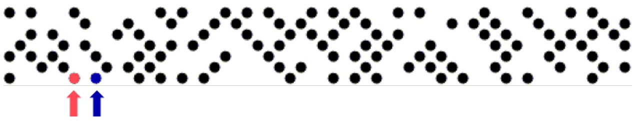

Figure 9. (Color online) Sample symbol showing one additional dot (dark blue dot) and one erasure (red dot) within the first codeword. The use of red dots was because it was determined the scanners filtered out red, which resulted in the red dots appearing as white to the scanners while allowing the researcher to still see the deleted dot

As previously discussed, the illumination of the corner dots was the prime consideration in symbol selection. So as to provide additional data points, symbols with one, two and three errors to the encodation pattern were included. Table 1, shows the various combinations of errors introduced to the symbols for testing.

Table 1. Damage test matrix. "Add" indicates that a dot was added and "Del" indicates that a dot was removed

	Codeword Damage					
	1st		2nd		3rd	
Run	Del	Add	Del	Add	Del	Add
1	Baseline - No Damage					
2	X					
3		X				
4	X	X				
5	X	X		X		
6	X	X	X	X		X
7	X	X	X	X	X	X

The simulator was run at 120 feet per minute to ensure sufficient decode time and nine hundred scans of each symbol for each of the two scanners used was undertaken. The data was captured and processed via Minitab. After completing the initial data run, which was used for an initial data analysis, additional runs were undertaken to allow for a fuller descriptive data analysis resulting in an overall data set that exceeded 12.6 million scans. This additional data was left out of the initial analysis so as to prevent the overall power of the analysis from becoming too great [10,11].

4. Results

An analysis of variance for the initial tests showed that there were statistically significant differences for the scanned bar code, F(28, 12024) = 4.30, p = 0.00, the level of damage, F(6,12024) = 29.36, p = 0.00, and the reader, F(1, 12024) = 722.46, p = 0.00, with no statistically significant grouping.

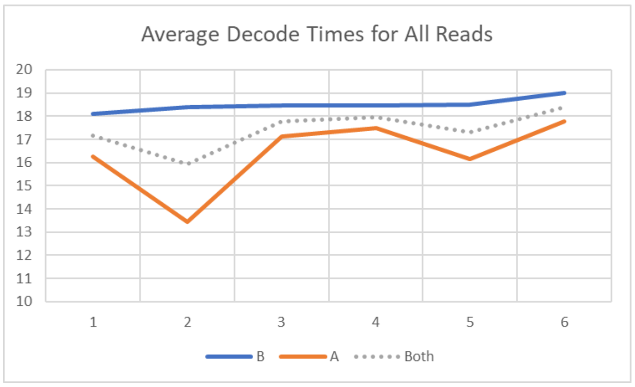

Figure 10. (Color online) Average scanner decode times for each damage run. Decode times are in milliseconds

Upon completion of the initial data runs, the extended data runs were executed and the overall aggregate results were compiled. As can be seen in Figure 10, a dip in the decode time for run "2" on scanner "A" occurred. The cause of this is not known, although this dip occurred after repeated attempts at ensuring that the scanner was correctly adjusted for both runs "1" and "2". Additionally, the generally upward trend noticed for reader "A" was a repeatable occurrence. Discussions with the manufacturer were not able to resolve the cause of this trend, as they believed that it shouldn't exist. However, since the results were repeatable, it has been assumed that the cause was either due to a) optimization routines within the reader that were not being disclosed or b) that there were slight imperfections in the setup and configuration of the reader (e.g. focus point, alignment, etc.) that were somehow impacting the performance of the reader when reading that specific set of symbols.

In addition to the odd dip in Figure 10, Figure 11, and Table 2, show that the consistency of performance of reader "B" was much better than reader "A". Once again, it was assumed that much of the inconsistency was due to the configuration and setup of the test rig and was not indicative of the scanner's true performance potential. Table 2, shows the overall descriptive statistics for the larger data sets. The variations in total bar codes scanned were due a limited time frame for which the bar code scanners could be used as they were needed for use in an academic class. As the smallest data run was for over 440 thousand scans, it was felt that the additional data points would not provide a significant difference in the results.

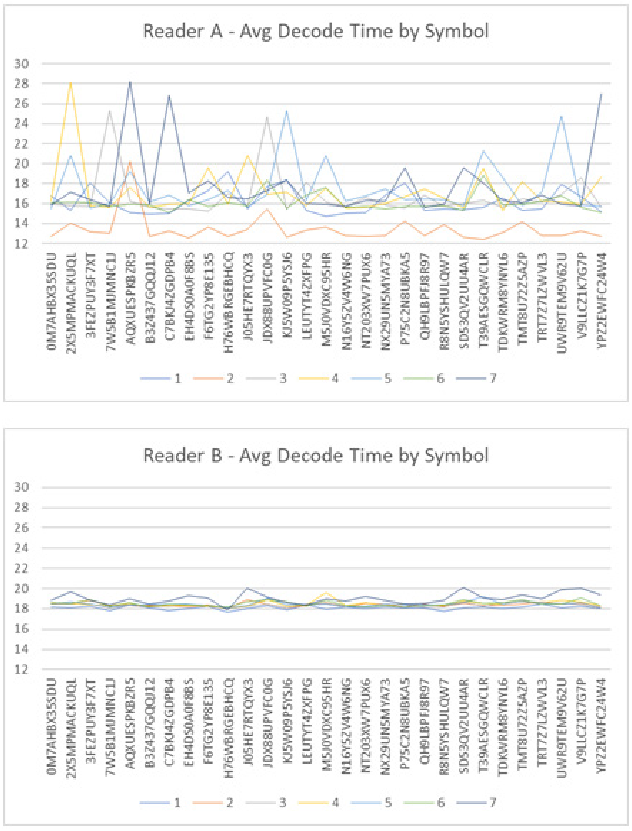

Figure 11. (Color online) Average decode by scanner and symbol for each damage run. Decode times are in milliseconds

From a practical viewpoint, the existence of a statistical significance between the various error levels can most likely be ignored. The lack of any statistically significant grouping during the initial post hoc analysis further strengthens this argument. As can be seen in the overall descriptive statistics, scanner "A" shows a 4.1 millisecond average range and scanner "B" a mere 1.2 millisecond average range. In either case even at speeds of 30 scans per second, the scanner has 33 milliseconds with which to find and decode a bar code. At 18.1 milliseconds, the extra 4.1 milliseconds would not necessarily prevent an undue number of reads to fail due to insufficient time.

Table 2. Overall descriptive statistics results by scanner.

Scanner A							
Run	1	2	3	4	5	6	7
Mean	16.6	14.0	17.3	17.8	17.8	16.4	18.1
Median	12	13	16	16	16	16	16
Mode	12	11	14	14	14	16	14
Std Dev	10.8	6.4	8.8	9.4	8.4	6.7	8.7
Kurtosis	11.2	28.8	25.1	22.4	20.2	41.7	18.8
Skewness	3.2	4.9	4.9	4.5	4.2	6.0	4.0
Min Time	10	9	11	13	12	12	12
Max Time	75	71	82	82	82	82	84
Total Reads	1,000,003	1,018,411	1,000,001	834,011	1,000,002	443,444	833,994
Scanner B							
Run	1	2	3	4	5	6	7
Mean	18.1	18.7	18.3	17.8	18.4	18.5	18.9
Median	18	19	18	18	18	18	18
Mode	18	19	18	18	18	18	18
Std Dev	0.5	1.9	0.8	3.5	0.9	1.0	1.8
Kurtosis	69.7	72.1	216.7	20.9	128.0	37.6	31.8
Skewness	-1.4	-6.4	-8.0	-4.6	-4.0	0.5	-2.3
Min Time	10	9	11	13	12	12	12
Max Time	75	71	82	82	82	82	84
Total Reads	1,000,003	1,018,411	1,000,001	834,011	1,000,002	443,444	833,994

5. Conclusion

While it may be assumed by bar code experts that the intentional inclusion of RSEC errors will have at most a marginal impact on overall performance, this study has shown that from an applied point of view, the impact of intentionally added errors to the DotCode symbology will result in, on average an extra 4.1 milliseconds to the decode time. And while the inclusion of intentional errors into a bar code symbol is never ideal, the added security of ensuring that a symbol is not inadvertently generated without a sufficient structure that ensures decoding can be easily argued. However, this conclusion needs to be embraced with caution as the one thing that this study did not attempt to address is the impact on decode time of errors in instances where a significant portion of the error correction code words were consumed. In instances where other damage to the dot code stream has occurred naturally (e.g. damaged labels, partially covered bar codes, etc.), the addition of intentionally induced errors could potentially result in significantly higher decode times or failed decodes.

Acknowledgements

This work would not have been possible without the support of Cognex, Microscan and Seagull Scientific. Cognex and Microscan both donated bar code scanners. Bartender Designer 2016 R3 was donated by Seagull Scientific. Additionally, the advice of the members of the AIM Technical Symbology Committee (www.aimglobal.org) was invaluable in both the design and analysis of the results. Further help was provided by Mr. Terry Burton. His encoder can be found at bwipp.terryburton.co.uk/. Finally, the personal assistance of Dr. Andrew Longacre was invaluable in understanding how the symbology worked and in how to best design some of the test symbols.

References

[1] AIM Global, Information technology — Automatic identification and data capture techniques — Bar code symbology specification — DotCode - Public Review, Cranberry Township, PA: AIM, Inc, 2017.

[2] Burton, T. Proposed "Mask Score, Print Quality Parameter", 2017.

[3] AIM Global, TSC 1709-29-Min-TSC DotCode Public Review Meeting_Sep17.

[4] Cognex, "The Rise of DotCode," Cognex, 26 August 2014. [Online]. Available: https://manateeworks.com/blog.[Accessed 21 February 2018].

[5] Dean, K. *GS1 DotCode Impact Assesment*. Inexto SA, Lausanne, Switzerland, 2017.

[6] Holliday, D. "Traceability Coding for Tobacco Products," 21 January 2016. [Online]. Available: http://www.labelingnews.com/2016/01/traceability-coding-for-tobacco-products/. [Accessed 21 February 2018].

[7] Nachtrieb, J. "DotCode: A real solution to a real problem - Barcode Test," Barcode-test, 1 August 2017. [Online]. Available: http://barcode-test.com/dotcode-real-solution-real-problem/. [Accessed 21 February 2018].

[8] Reynolds, P. "Walmart roils the case-coding waters," 26 April 2016. [Online]. Available: https://www.packworld.com/print/63634. [Accessed 1 March 2018].

[9] Rittenburg, J. "Safety in Numbers," *Pharmaceutical Executive*, 26 (6) pp. 46-50, June 2006.

[10] Baguley, T. "Understanding statistical power in the context of applied research," *Applied Ergonomics*, 35 (2), pp. 73-80, 2004.

[11] Lin, M., Lucas, H. C. and Shmueli, G. "Too Big to Fail: Large Samples and the p-value problem," *Information Systems Research*, 24 (4), pp. 906 - 917, 2013.