Long term digital document survival using open source applications and operating systems

Sherine M. Antoun

University of Wollongong, sherine@uow.edu.au

Recommended Citation


Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au
NOTE

This online version of the thesis may have different page formatting and pagination from the paper copy held in the University of Wollongong Library.

UNIVERSITY OF WOLLONGONG

COPYRIGHT WARNING

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.
Abstract

The history of early materials is one of loss and eventual partial reconstruction through fragments. Today's digital document is not immune from loss, but whereas early material was capable of being partially recovered, a bit stream of zeros and ones cannot be intelligibly recovered from fragments!

The research seeks to investigate a contingency of last resort; by providing current viewing tools adaptable to future systems for future users to display the stored digital documents, when other archiving methods fail. This approach is designed to be non-restrictive, allowing the use of any authoring tool available at the time to create digital documents. The research describes a viable method for archival storage, and a viable approach to dealing with issues of viewing and authoring software obsolescence.

The capacity of Open Source applications to transcend the divide between operating platforms is demonstrated by this research. It argues the viability of Open Source applications in preserving data for medium to long term based on the adaptability of the software, the source code being stored along with the data. Where the operating environment evolves to become unsuitable to the application, the first option is for the source to be edited / ported to the new environment. The alternate option is to develop a system compatible compiler to compile the stored source code. It is worth noting that both options require a non-trivial effort on the part of future users; however, option two is a once only effort as the compiler can be used repeatedly to compile multiple applications.

The viability and flexibility of the Open Source approach to archival survivability is also demonstrated by this research, the users are afforded maximum flexibility as to file type and application of choice, in contrast to other approaches such as Victorian Electronic Records Strategy (VERS) that restrict the users to a limited pool of case specific tools to use.
Acknowledgements

This thesis grew out of an interest sparked in me by my Supervisors Professors Carol Alcock and John Fulcher in the field of long-term data preservations. Their comments on chapter drafts were instrumental in teaching me critical thought upon which I will always draw. I would like to thank them both. I could not have imagined having better advisors and mentors for my research, and without their common sense, vast knowledge, perceptiveness, patience, and cracking-of-the-whip I would never have finished. Amid the chapter, drafts and redrafts both were always around to offer quiet encouragement and extraordinary kindness. I am very grateful for my years at the University of Wollongong. I am also indebted to all my other Professors at the School of Information Technology and Computer Science for their advice and comments on the proposed research and methodology and all other productive comments throughout my candidature.

Finally, I wish to thank my wife Elena and children Annette Diana and Alexander for their unwavering support of my academic endeavours over the past few years.
# Table of contents

Long Term Digital Document Survival Using Open Source Applications and Operating Systems .................................................. 1  
Master of Information and Technology (Research) .................................................. 1  
Sherine Micheal Antoun ............................................................................... 1  
Declaration ............................................................................................... i  
Abstract .................................................................................................... ii  
Publications from this Thesis ...................................................................... iii  
Acknowledgements ...................................................................................... iv  
Table of contents ......................................................................................... v  
Table Of Figures .......................................................................................... ix  
Table of Tables ............................................................................................. x  
Chapter 1 .................................................................................................... 1  
  1.1 Introduction – Statement of the Problem .................................................. 1  
  1.2 Background to the Problem .................................................................... 3  
  1.3 Aim and objectives of this thesis ............................................................... 5  
  1.4 Scope of this thesis ................................................................................. 7  
  1.6 Assumptions and Limitations of this Thesis ............................................. 8  
  1.7 Methodology Used .................................................................................. 9  
  1.8 Justification and Importance of the Study .............................................. 9  
  1.9 Outline of the Following Chapters ........................................................... 10  
Chapter 2 – LITERATURE REVIEW ............................................................ 12  
Long-term data storage and retrieval .............................................................. 12  
  2.1 Overview ................................................................................................ 12  
  2.2 Towards Archival Document Survival .................................................... 13  
    2.2.1 What is archival document survival .................................................. 13  
    2.2.2 Media Longevity ............................................................................. 14  
  2.3 Modern storage technologies .................................................................. 18  
    2.3.1 Redundant Arrays of Inexpensive Disks (RAID) ................................. 18  
      2.3.1.1 Will Proprietary RAID Transcend the Passage of Time? .............. 18  
    2.3.2 Genomic Storage technology ............................................................. 20  
    2.3.3 Refreshing and Conversion ............................................................... 22  
      2.3.3.1 Migration ................................................................................. 24  
      A - Operating System preservation ....................................................... 25  
      B - Hardware preservation .................................................................. 25  
      C - Simulating Operating System ......................................................... 26  
      D - Simulating Machine Behaviour ....................................................... 26  
    2.3.5 Oltmans' E-Depot ........................................................................... 26  
  2.4 Preservation Trends ............................................................................... 28  
    2.4.1 Emulators, Simulators and Virtual Machines ................................. 28  
  2.5 Latest preservation technologies ............................................................. 32  
    2.5.1 Buckets ....................................................................................... 32  
      2.5.1.1 Why not buckets? ................................................................... 33  
    2.5.2 VERS ....................................................................................... 33  
  2.6 Preservation trends .................................................................................. 35  
  2.7 Removable media and long-term storage? ......................................... 36  
  2.8 Future Digital Archaeology .................................................................... 38
5. Discussion ................................................................. 143
6. Conclusion ................................................................. 144
Bibliography ................................................................. 145
Table Of Figures

Figure 2.1- Magnetic disk damage (diagonal cracks) data is retrievable using Magnetic force Microscopy, interpreting the retrieved binaries however is not a trivial task. (Ross, 1999, p33) ................................................................. 15
Figure 2.2 Grooving: fungus spores bore through a CD (Bosh, 2001) .................. 16
Figure 2.3 A recumbent plasmid with 2 DNA fragments as sentinels (Codons) protecting the encoded message in between (after Wong, 2003, p97). .......... 21
Figure 2.4 Removable media expected life span (Gilbert, 2003, p2) .................... 37
Figure 2.5” the medium is a short-lived message” (Rothenberg, 1999, p3) .......... 38
Figure 3.1 Network diagram (Java Lab at SITACS within the University of Wollongong) ........................................................................................................ 45
Figure 3.2 “UI’ OpenOffice “Write” on Linux Mandrake 9 and MS office “Word” on Windows 2000 ......................................................................................... 54
Figure 3.3 “UI” OpenOffice “Calc” on Linux Mandrake 9 and MS office “Excel” on Windows 2000 ......................................................................................... 54
Figure 4.1 Error encountered when attempting to use a 3.5 inch magnetic disk unused for a few years ................................................................. 66
Figure 4.2: Effects of 3% artificial noise ............................................................ 79
Figure 4.3: Effects of 7% artificial noise ............................................................ 80
Figure 4.4: Effects of 11% artificial noise .......................................................... 81
Figure 4.5: Effects of 18% artificial noise ......................................................... 81
Figure 1 – Java Laboratory (of 36 SUN Workstations) ...................................... 136
Figure 2 – Effects of 3% Introduced Noise ...................................................... 141
Figure 3 – Effects of 18% Introduced Noise .................................................... 142
Table of Tables

Table 3.1 Comparison of Office suites ................................................................. 57
Table 4.2 Percentage of damage observable at a 3% level of artificially introduced noise................................................................. 77
Table 4.3 Percentage of damage observable at a 7% level of artificially introduced noise................................................................. 77
Table 4.4 Percentage of damage observable at an 11% level of artificially introduced noise................................................................. 78
Table 4.5 Percentage of damage observable at an 18% level of artificially introduced noise................................................................. 78
Chapter 1

1.1 Introduction – Statement of the Problem

The preservation of digital data problem originally appeared in the pre-electronic age, with the invention of sound recording. Signals were recorded on medium that required a specific machine to render it back into a form that could be apprehended by the senses (Gruber, 1993, p1). Then, there were dozens of incompatible recording formats. Eventually the 78 Revolutions per Minute (RPM) became widespread and eventually the standard of choice for audio recording publishers, but not before other formats had produced a substantial body of recorded material, some of it irreplaceable. Today serious audiophiles have constructed customised equipment that could play back Edison cylinders, various disk formats, those that ran from the centre axis to the circumference, and the standard circumference to centre axis disk with their varying speeds. Decades passed between one audio recording standard and another (Gruber, 1993, p1).

The advent of digital computing in the early '50s vastly accelerated the pace at which formats designed to store information evolved and became obsolete. As computers increased an order of magnitude in speed every two or three years and decreased in cost, the pressure to replace old, less efficient standards with newer, faster, more efficient peripherals became irresistible (Gruber, 1993, p1).

Digital information - a large part of the current era's cultural and intellectual heritage - offers significant benefits to its users. Preservation of and access to this information is, however, dependent on impermanent media, evolving technologies, context retention, and copyright.
Digital data reproducibility gives users a false sense of security as it makes it theoretically invulnerable to the ravages of time; however, the physical media on which it is stored are far from eternal. Optical CDs, DVDs, and magnetic disks all have finite serviceable life span, attempting to read magnetic media often just a few years after it was used last often is a waste of time.

The contents of most digital media become irretrievably corrupted long before words written on paper and the technology behind them often become obsolete much sooner. Likewise, format compatibility becomes an issue with digital document retrieval within a short period of authoring (Rothenberg, 1999, p4). "Digital information lasts forever or five years, whichever comes first." (Rothenberg, 1999, p4).

As authoring tool technologies evolve, older tools become obsolete and fade into disuse as do their associated viewers. Reliable document archival and retrieval is essential for business, government, and scientific endeavour. Irretrievable archives are an impediment to good business and to progress in general.

With authoring and viewing tool obsolescence, future generations run the risk of losing the archived legacy of the current one. The task of preserving and disseminating today's business and intellectual output is by far the most serious challenge of the information age.

Previous civilizations had the enviable luxury of stable media, limited data volumes, and long lived languages to use. By contrast, today, more information is being produced, archived, and preserved in fewer libraries, with each library having fewer resources (Nelson, 2000, p369).
The ability for long-term storage, and retrieval, of the terabytes of data that various disciplines of scientific research produce today is questionable at best, as is the reliability of commercial methods employed by business to archive the petabytes of data produced there.

This problem is further compounded by the constant changes to digital document formats arising from software version changes, and eventually by the obsolescence of the authoring software. It is also worthy of mention that hardware changes over time also lead to a point where the actual storage media are no longer supported (Rothenberg 1999, p19). In short, the challenge is to combat entropy to prevent ordered systems from descending into chaos.

1.2 Background to the Problem

There is a large body of work being done on archival longevity worldwide of which six are at the forefront of archival longevity research. The first is Ensuring Longevity of Digital Document by Jeff Rothenberg (1999). The second work is An Experiment in Using Emulation to Preserve Digital Publications by Jeff Rothenberg (2000). The third is Long Term Preservation Of Digital Publications by Lorie (2000). The fourth is Digital Archaeology by Seamus Ross (1999). The fifth is Preserving Digital Information Forever by Andrew Waugh (2001), and the sixth is Preserving Functionality in a Digital Archive by Eric Oltmans (2004). This body of work has identified many problems in the area of long-term digital document storage; a great deal more work is needed to solve these problems however. Many solutions have been suggested. These include procedural algorithms such as data migration, original software system archival, and system / software emulators. Reviewing the published
scientific work in the field of digital archival longevity herein discussed in chapter two has revealed that a single solution to the archival problem remains elusive; at best an acceptable compromise can be found between longevity of the digital document archive and lossiness of the archival algorithm. Simply stated, long term digital archiving is the art of storing unstable electronic bit patterns, on unstable media, to be retrieved by equipment fast becoming obsolete, for periods of time that vastly exceed the media expected life span, while expecting lossless recovery.

The pervasive use of digital technology to produce documents, databases, and publications has a serious flaw: there are, so far, no available proven techniques for ensuring that digital information will remain accessible, readable, and usable in the distant future. Software developers provide migration tools for documents from one generation to the next but as developers evanesce from the information technology landscape, competitors do not always provide such migration tools. These migration paths involve transforming the old files into the newer format and saving them on more modern storage media, a task which is both time consuming and costly for organisations with large data archives. More importantly, the migration process requires a conscious act on the part of the users. Time and financial constraints often cause the deferral of such tasks to a point in time when the old records deteriorate and become irretrievably lost. (Rothenberg 1999, p17) Coupled with the time and cost involved in migrating digital data from one format to the next, the process involves physical changes to the data stored (e.g. binary representation / layout / encoding etc...) from the older format to the newer, the cumulative effect of repeated changes results in a degradation of the evidentiary status / value of the digital document(s) (Thibodaux 2001, 7 & 14).
1.3 Aim and objectives of this thesis

Business organisations and individuals are often faced with format and application obsolescence issues when dealing with archival documents. Documents that have survived the passage of time on storage media become inaccessible due to application obsolescence. Users are then left with the option of re-installing old applications on new systems; this, however, is not always possible due to incompatibilities. The aim of this thesis is to explore a robust, low cost, low overhead, and widely available method for long term digital document preservation that is capable of transcending system and software application obsolescence, while remaining capable of delivering to future users the digital documents stored as their original authors saw them. To effectively archive a document long term, it must appear to the user at the point of retrieval as its author saw it, namely: it must retain functionality, it must retain content integrity, it must be editable, and it must retain the ability to be saved back to its original format.

This research provides a means for long-term digital archival success using freely available tools, Open Source systems, and simple methods that require minimum intervention. Further analysis leads to determining criteria for measuring the success or otherwise of the archival process.

Digital information is increasingly becoming the predominant output of the current era's cultural and intellectual heritage offering significant benefits to users. At the same time preservation and access to this information is dependent on impermanent media, proprietary software, changing technologies, context retention, and copyright protection. Future generations are at risk of losing the legacy of the current one. The task of preserving and disseminating today's intellectual output is by far the most
serious challenge of the information age; previous civilizations had the enviable advantage of stable media, long-lived languages, and relatively limited output storage material. Many schemes, software tools, and hardware solutions are offered as a means to store digital documents, however thus far, the goal remains elusive. This thesis describes a low cost, low overhead, and widely available method for long term digital document preservation that is capable of transcending system and software application obsolescence while remaining capable of delivering to future users the digital documents stored as their original authors saw them. The following objectives clarify the path taken to achieve this aim.

1. Examine the methods, mechanism, and technologies used today for preserving digital documents.

2. Understand how each provides for long-term digital document survival and identify shortcomings if any for each method.

3. Devise a low cost, low overhead, and widely available method for long term digital document preservation that is capable of transcending system and software application obsolescence while remaining capable of delivering to future users the digital documents stored as their original authors saw them.

4. Apply knowledge gained from the literature review in designing a suitable experiment to test the validity of the proposed archival method.

5. Evaluate if the devised archival method is a suitable solution for the long-term digital document survival issue.
1.4 Scope of this thesis

In looking for a solution to the problem of digital obsolescence this research thesis focuses on the examination of Open Source systems and tools as a viable mechanism for delivering current digital output to future users free of the limitations other mechanisms suffer from. The literature, reports, and research publications utilised in this study come from a variety of sources, and have varied aims and applicability. It is worth noting however that all are focused on the archival and subsequent successful retrieval of digital documents. The conditions in these differing studies vary according to the circumstances of work they describe but most have similar goals.

The software used and mentioned in this thesis is all Open Source. The authoring software used to write this very report was a combination of Open Source and proprietary software, namely Open Office and Microsoft Office, with the bulk of the editing done using Open Office. This software is freely available under the General Public License (GPL) agreement and can be downloaded from the internet; the primary download site for this experiment was Source Forge (www.sourceforge.com).

An attempt to search for information as to which is the best distribution of Linux proved futile as each distribution had its own advantages and peculiarities. Trial and error in the case of this research revealed that for use in conjunction with the experimental equipment described in 4.1 Appendix iii; Mandrake9.0 was the most compatible distribution supporting all the hardware devices (sound device, graphics device, removable Compact Disk / DVD Drive, removable zip drive, removable floppy disk drive, and wireless network card). Mandrake 9.0 is available for download from http://www.mandrake.com.

Detailed examination of software costs and equipment involved with the experiment
will not be included as the open source software was all free; hardware costs constantly change. Moreover, the range of products available is extensive and would require a separate research report to list all options / vendors combinations. The network designs contained in this report were current at the time of writing. All products mentioned in this thesis are current (as at June 2004) and remain freely available under the GPL system.

1.6 Assumptions and Limitations of this Thesis

Archival arrangements within organisations differ depending on needs and available infrastructures, the archival processes must always follow set down procedures utilising personnel media and equipment. As resources are finite, organisations assess the storage capacity and evaluate the needs for archival survival beyond the life expectancy of the current infrastructure. The thesis describes an experiment seeking to ascertain how documents produced using proprietary software can be accessed, modified and saved by Open Source software tools run within an Open Source operating environment.

The primary focus of this research is on the usability of Open Source tools and operating environments as a vehicle for long-term digital archival survival. The reason for this focus is the belief that this approach offers the greatest chance for digital documents to transcend the life and eventual obsolescence of the proprietary software used to author them. This approach also is least restrictive to authors of digital documents as they are not constrained by the range of authoring tools or format prescribed by other archival survival approaches.
1.7 Methodology Used

The primary method of research was the experiment described in Chapter 3. This consisted of a number of iterations of the experimental process in order to gather data. The process involved storing a collection of digital documents on local unused disk space within individual nodes of an experimental network cluster, thence subjecting the collection to varying amounts of introduced noise to simulate the effect of the passage of time, and finally attempting to view the documents using Open Source software applications while assessing the quality of the recovered documents using metrics to determine the amount of degradation the documents suffered. The data gathered is analysed and inferences drawn from these experimental results. The combination of literature review and cyclic experimental process was used as this best catered to the needs of this research project as explained later in Chapter 3. The analysis of the research culminated in the findings described in Chapter 4. No other method would yield as accurate or effective results for this area of research.

1.8 Justification and Importance of the Study

A thorough examination of the published scientific work could not identify work(s), which assesses the applicability of Open Source operating environment and software tools to long-term digital archive survival. Nevertheless, this remains the one example in modern day computing endeavours, where both the software tool and the technology behind the tool are available for both everyday use and study purposes. Proprietary software is only available under licence and then only in executable format. It is dependent on hardware and operating system compatibility; Open Source is available as both executable, and source code. It is the source code availability that
enables Open Source tools to transcend system dependability, since users are allowed to, often encouraged to; port the tools to their own computer system platforms.

Digital archives can be evidence of business transactions, authors’ intellectual output, users’ personal communications, and, more importantly, this civilisation’s legacy. The preservation of such documents is an effort that should be shared among the authors and the readers. Just as today’s historian invests time and effort to read the legacy of past civilisations, future users will need to invest time and effort to read this civilisation’s legacy. The experiment this thesis describes preserves the technology future users may need to become familiar with in order to be able to access the preserved data. Utilising the millennia old principles of the Rosetta stone, this experiment describes the means to preserve the data along with the technology originally used to create that data.

1.9 Outline of the Following Chapters

Chapter 2 - Literature Review: During this research project, an extensive literature review was undertaken. The findings of this literature review are presented in this chapter, beginning with an examination of current processes and schemes such as archival refreshing, format migration, and emulation. This served as the premise for subsequent development of the experimental methodology.

Chapter 3 - Research Method: Design Methodology Used In This Study: In this chapter the methods and processes used in this research are explained and justified. The technique adopted here includes a number of iterations of the experimental process briefly outlined in 1.8; the reasons for using this approach are outlined, along
with a description of the software tool used to assist the process.

Chapter 4 - Research Results Analysis: Results gathered from the experimental cycles are summarised in this chapter.

Chapter 5 - Research Discussion: A careful evaluation of these results outlined in chapter 4, along with other knowledge, is used to draw inferences and arrive at the conclusions reported in chapter six.

Chapter 6: In this final chapter, the objectives are reviewed with concluding notes made on what was demonstrated during the course of this research project. Recommendations based on knowledge and findings gained are made concerning the future of long-term digital document survival. Lastly, overall concluding remarks are made regarding the research topic.
Chapter 2 – LITERATURE REVIEW

Long-term data storage and retrieval

2.1 Overview
While it is possible to read the 400-year-old books printed by Gutenberg, it is often difficult to read a 15-year-old computer disk (Kahle, 1996). The task of preserving and disseminating today’s intellectual output is by far the most serious challenge of the information age; previous civilizations had the enviable luxury of stable media and long lived language. The preservation processes are diverse, reliant on periodic user actions such as physical data migration, and refreshing. This literature review provides a comprehensive overview of the major directions the archiving of digital documents effort has taken. There are various approaches, process and concepts that exist in this field each with a valid claim as to its effectiveness as a long term preservation process, these vary from Rothenberg’s approach of emulating an operating system on another (2000), to limiting the users to a small pool of authoring applications as Waugh describes in the Victorian Electronic Records Strategy (VERS) (2001), to development of an archival model and a purposed built document retrieval systems such Nelson’s Buckets approach based on the Smart Object Dumb Archive. Devised for NASA (2001) or Oltmans approach of preserving functionality within the digital archive (2004). This examination will demonstrate the need for a different more independent archival approach free from constraint of operating environment or hardware survival that characterise the current approaches.
2.2 Towards Archival Document Survival

2.2.1 What is archival document survival

"The fundamental principle behind any digital preservation strategy must be 'do minimal harm'." (Waugh, 2000, p176) The survival of a digital document encompasses: physical survival, functional survival, and content survival. Simply put it is the ability to retrieve an archive and for it to be appear at the point of retrieval as its author saw it at the time of creation. Beagrie defines long-term preservation as meaning two distinct but equally important functions: long-term maintenance of a byte stream and the ongoing access to its contents through time and changing technology (Beagrie, 2001, p5).

The first and simplest approach to preservation is the physical preservation of the bit stream against the obsolescence of media and/or device. This approach involves the use of modern storage technology to maintain a viable digital archive over an extended period, using such mechanisms as Redundant Arrays of Inexpensive Disks (RAID). A second approach uses Genomic Storage technology; which involves using redundant genetic sequences in a living organism (Wong, 2003). A third approach involves conversion or migration, which has been used for decades to preserve operational data in data processing installations (Lorie, 2001, p347). The migration process requires a conscious effort from the users; it is a time consuming costly process, and migration degrades the quality of a digital document as it involves physical changes to its contents. Migration is defined as "the periodic transfer of digital material from one hardware/software configuration to another, or from one generation of computer technology to a subsequent generation" (Muir, 2001, p169).

A fourth approach involves emulation. Emulation is a widely used technique, in
which one computer system reproduces the behaviour of another system, and thus the emulator can run software designed for the system it is emulating (Rothenberg, 2000, p5). The fifth approach attempts to optimise the chances of document survival by adopting an array of approaches distilled from legacy proven preservation methods each with its own advantages and drawbacks (Oltmans, 2004).

The efforts to preserve digital documents are numerous and varied. Each of these approaches is based on an application of one or more technology, but for emulation, all share a common thread that is preservation based on technological advancement. These approaches, as will be discussed later on in this chapter, are not without risks. Some rely on technology that will inevitably become obsolete, others rely on technology that requires a sophisticated retrieval process along with a high degree of content knowledge, and some simply modify existing documents to suit new software applications. While leading edge technology is being employed to produce new digital documents, it is trailing edge technology that can access and display archival digital documents produced by software applications that have been superseded by current technology. Emulation on the other hand requires extensive knowledge of obsolete, hardware architecture, operating environments, and applications design, in order to build, within a modern system environment, a virtual machine capable of running obsolete applications to access archival digital documents.

### 2.2.2 Media Longevity

Today, digital media is fast becoming the media of choice and unlike granite, it is rather fragile. While storing and replicating data on digital media are simple processes, retrieving data from old digital media may be hampered by one or more
obstacles such as:

(i) Media degradation resulting from storage in conditions where temperatures, and / or relative humidity, are less than ideal.

(ii) Distortion in the case of magnetic media where media come into contact with, or is influenced by, magnetic materials, or physical damage as in scratches / cracks in the case of optical media.

(iii) Optical disks suffer from the oxidisation of the reflective material inside the CD. CD-ROM (data or audio) is a 1.2mm disc of polycarbonate pressed with a spiral pattern of pits on one side – which is sprayed with a very thin reflective layer of aluminium, then sealed with a thin layer of lacquer and printed with a decorative or descriptive label. The problem is the gradual oxidisation of the reflective material that lets the laser read the pits embedded in the CD. This happens through the label side where the reflective aluminium is only protected by a relatively thin layer of lacquer. Printing inks used by pressing plants react with the lacquer, the protective coating is compromised and the reflective layer is then

Figure 2.1- Magnetic disk damage (diagonal cracks) data is retrievable using Magnetic force Microscopy, interpreting the retrieved binaries however is not a trivial task. (Ross, 1999, p33)
exposed to humidity and oxidisation takes place. In climates that are warm and humid CDs, develop noticeable clumps of a white or grey spreading fungus on either or both surfaces. Some are subject to noticeable fungal infection spreading into the reflective layer from the centre as well as from the outer edge. The fungus grows in the area between the lacquer and the polycarbonate, after gaining entry through poorly sealed edges or through holes in the lacquer layer. The fungus actually eats away at the pits in the polycarbonate and allows faster oxidisation of the aluminium – rendering the affected areas unreadable. The organic matter of CDs was found palatable by such an organism. Damage by grooving fungus is slower than by oxidisation; however, it eventually renders affected CDs unusable (Bosh, 2001).

![Figure 2.2 Grooving: fungus spores bore through a CD (Bosh, 2001)](image)

- (iv) Disaster such as lightning strikes, fires and so on.
- (v) Wear and tear because of excessive use especially in the case of punched paper tape.
- (vi) Manufacturing defects of all types of media.
- (vii) Loss of functionality of access devices resulting from technological obsolescence or components in mechanical device wear.
An increasingly shortened serviceable life expectancy of hardware; in other words mass manufactured tape devices are made using components built to budget rather than quality. While these devices function, well they have a limited serviceable life expectancy. Added to this is the fact that older hardware devices and drivers are not supported by newer hardware beyond one or two generations.

Loss of manipulation capabilities due to changes in hardware and operating systems rendering it impossible for applications to perform the same functions or access the same data manipulation routines.

Loss of presentation capabilities with changing video display technologies, as demonstrated by the rapid obsolescence of computer games. These are heavily reliant on the ability to address the video drivers directly, as graphic hardware and driver technology of older games become incompatible with newer technology.

Weak links in the creation, storage, and documentation chain; media remains readable but unintelligible as the encoding strategy cannot be identified. This is often caused by the loss of documentation relating to the encryption algorithms and keys (Ross, 1999, pp3-6).

Today, more than ever before, the effort and indeed the goal is to attain an acceptable level of survival of digital documents, which encompasses: physical survival, functional survival, and content survival.
2.3 Modern storage technologies

This section considers a selection of the preservation efforts recently proposed as outlined in Section 2.2.1. It outlines their basic concepts and the reasons for not considering them for the purpose of this research.

2.3.1 Redundant Arrays of Inexpensive Disks (RAID)

There are many good reasons for considering RAID as a long-term storage vehicle. A few are: the ability to combine several physical disks into one larger “virtual” device, performance improvements, and the many forms of redundancy built into the architecture. RAID joins multiple drives into a single block device where data is broken down into blocks and each block is written to separate disk drives. Performance is greatly improved by spreading the I/O load across many channels and drives; data is retrieved in a similar manner (Raid, 2001, p2).

Often RAID is employed as a solution to storage and performance problems. While RAID is indeed often the solution sought, it is not a silver bullet. RAID is a complex software and hardware device that is in constant use. Disk swaps and redundant disk and data render it ideal for long term data archiving, RAID deals effectively with cache coherency issues associated with multiple redundant back ups. (Raid, 2001, p1)

2.3.1.1 Will Proprietary RAID Transcend the Passage of Time?

RAID systems are proprietary systems and as such are susceptible to obsolescence
issues, a proprietary application relies on its vendor’s survival for longevity, as vendors succumb to market forces and disappear either by going out of business or by being taken over by other vendors. The unpublished body of their proprietary design documentations and source code renders their obsolete technologies unmaintainable within a changing hardware and software technology environment. This means that when a commercial RAID becomes obsolete after the demise of its vendor its contents become lost. The only exception to the “trade secret” is the Linux software RAID, the source code of the Linux software Raid is the only freely available RAID source code, however it is but one of a number of available RAID (Raid, 2001, p1). Furthermore, the Linux software RAID cannot be adapted to operate commercial RAID silos as these are purpose build proprietary machines with specialist drivers and software interfaces that are part of the trade secret that disappears along with the system vendors.

As RAID is a hardware and software device, it is susceptible to hardware failures and device obsolescence thus endangering the long-term survival of data archived on the system. There is no guarantee that hardware devices will remain serviceable in the distant future, and with the evolution of technology, current hardware is unlikely to be maintained beyond a few years.

Aside from the substantial initial cost, RAID is also a resource intensive device requiring administration and upgrade, used to house data that has fallen out of everyday use; it is invariably, to many organisations, an expense for no expected return.
2.3.2 Genomic Storage technology

A new and long lasting medium for information storage currently under investigation is Deoxyribonucleic Acid (DNA), data memory technology, which has a life expectancy much greater than any existing electromechanical or optical media. The DNA memory prototype consists of four main steps:

a) Encoding meaningful information as artificial DNA sequences;

b) Transforming the sequences to living organisms;

c) Allowing the organisms to grow and multiply;

d) Extracting the information back from the organisms.

Genetic data storage on unused DNA sequences within a living genome is a viable possibility for digital storage. Because a DNA sequence is digital, it can be used to construct any English text, just as binary numbers 0 and 1 are used to encode ASCII characters. (Wong, 2003, p95) The genome of Deinococcus having already been completely sequenced is an ideal vehicle for this purpose. It comprises a set of fixed-size sequences (20-base-pair long in experiments) that do not exist in the candidate bacteria yet satisfy all the gonomic constraints and restrictions required for the purpose. (Wong, 2003, p97) Multiple triplets called “Stop Codons” exist within its genetic make up; telling the bacterium repeatedly it has reached the end of the native DNA sequence and should stop translating its contents. Without the protection of stop codons, the bacterium could misinterpret the encoded information and produce artificial proteins that could destroy the integrity of the embedded message or even kill the bacterium. (Wong, 2003, p98)
The DNA fragment is then cloned into a recombinant plasmid (a union of foreign DNA fragments into a circular DNA molecule). The vector and the encoded DNA are then incorporated into the genome of Deinococcus for permanent information storage and retrieval. Wong observed that deinococcus granted perfect protection for the embedded message, and reproduced well (Wong, 2003, p97).

DNA memory can be expanded dramatically by storing different pieces of information in a population of bacteria; considering that a millilitre of liquid can contain up to $10^9$ bacteria, the potential capacity of bacterial-based DNA memory is enormous, assuming a well-designed data index scheme. To deal with the potential of mutation in the organisms affecting the integrity of the embedded messages, the bacterium selected is characterised by a low mutation rate. However, random changes can still occur. The need for:

- a wet laboratory environment
- extensive supporting data as to the location, nature and most importantly
- contents of the genetically encoded message in order to retrieve it,
renders genomic data storage unviable for long term digital document archiving. The reason is there can be no guarantee as to the survival of the complementary data required for the retrieval effort (Wong, 2003, p98). The yet unproven reliability of Genomic storage, especially the undocumented effect of successive mutation of the bacterium used, detracts from the viability of this approach.

2.3.3 Refreshing and Conversion

Digital information is frequently copied to new storage media, to circumvent loss due to physical media deterioration or media and device obsolescence. This process is known as refreshing. Eventually, refreshed records encounter a more serious problem: namely, the application that created and permitted the viewing of these records as their original author saw them becomes obsolete.

As applications become obsolete, it is then that the digital archival records face a greater challenge. The fact that conversion / migration of digital documents from the original logical format into successive subsequent formats as each previous format becomes obsolete is problematic and labour-intensive (every document must be converted in this way every time its current logical format becomes obsolete.). The migration process is described in details later in section 2.3.3.1.

In reality the digital document faces “double jeopardy”: it has to survive on its storage medium and again it has to survive successive conversion efforts. The migration process changes the file from its existing format to a newer format; this process changes the document. The cumulative effect of successive migrations degrades the evidentiary value of the documents. Simply put, the document after several
successive migrations may bear little or no resemblance to the original as its author saw it. The reliability of the document is diminished (Muir, 2001, p174-5). From a legal standpoint the evidentiary value of a document changed from its original form is compromised rendering it unacceptable as legal evidence.

Many organizations manage their information poorly and a migration program may miss significant amounts of information (Waugh, 2000, p176). Migration costs money and the temptation in many organizations will be to delay implementing a migration program. Some organizations will delay too long and find their information cannot be converted in a cost effective way (Waugh, 2000, p177).

Another significant challenge is that migration may break the cardinal rule of preservation: minimize harm. As described below in section 2.3.3.1 migration explicitly means modifying the data, and this modification will degrade the preserved information especially if the new format cannot support aspects of the original format. Worse, it subsequently is impossible to determine what has been lost (Waugh, 2000, p176). Successive migrations may cause the data to be so degraded that it is effectively lost. Demonstrating that the migrated copy of the record is a true and accurate copy of the original may cause problems. Quality control and testing is a significant cost in any migration process (Waugh, 2000, p177).

Repeatedly migrating a document from one format to another introduces subtle changes that are not immediately obvious. The reason for this is that the target format is not 100% compatible with the source format otherwise migration is not required (conversion tools use methods such as heuristic match, approximation and similar to generated the new format). An extra white space character in a text document may be inconsequential. However, an extra coma in a coma-delimited spreadsheet could
throw the balance sheet into complete disarray. The cumulative effects of undetected (and even worse undocumented) subtleties of any format conversion process detract from the overall reliability and viability of conversion / migration as a long-term archival preservation process. While format conversion may not be a viable preservation mechanism, adapting modern equipment to old software (emulation) may be.

2.3.3.1 Migration

Migration is an active, system-based approach that alters the digital document, often involving schema conversion. This migration / conversion process involves data transformations that materialise the schema transformations required to render data files generated by legacy software accessible to modern software applications (Henrard, et al, 2002, p212). The conversion / migration processes may potentially degrade or corrupt the digital document, destroying its original appearance, structure, interactive behaviour, its’ look-and-feel’ and even its actual content (Waugh, 2000, p179, Rothenberg 2000, p12, Henrard, et al, 2002, p210). The effects of the migration tools on digital documents are documented by their vendors, however the successive / cumulative effect of repeated migration over time is undocumented.

2.3.4 Emulation

While refreshing and format migration provides short term answers to some challenges faced by ageing digital document archives, saving obsolete software to read old document formats is a more logical approach to preservation, as it does not involve repeated alteration to the documents as does migration. This approach, however, presents a new set of challenges, as software is always Operating System-
(OS) dependent, and may also be machine generation specific (Muir, 2001, p168).

This leads down several paths: preserving operating systems, preserving machines, simulating operating systems, and simulating machine behaviour.

A - Operating System preservation

Operating systems perform basic tasks, such as recognising input from the keyboard, sending output to the display screen, keeping track of files and directories on disk, and controlling peripheral devices such as disk drives, printers, and modems. Operating systems provide a software platform for application programs on which to run, and manage system resources. Application programs are compiled to executable code to run within a particular operating system environment. As operating systems interface with hardware as their primary function, they are inherently hardware specific. Preservation of operating systems is dependent on hardware preservation (Muir, 2001, p168; Rothenberg, 2000, p27).

B - Hardware preservation

Software is hardware dependent for input, output, and computation. The idea of being able to maintain hardware in working order forever is unrealistic; eventually mechanical devices will fail, circuit boards will fry, silicon chips will no longer function, and mechanical moving parts will succumb to corrosion or wear out. The cost of replacing these items in the distant future where no production facility is likely to remain will be a formidable deterrent (Muir, 2001, p169; Rothenberg, 1999, p2).
C - Simulating Operating System

The task of simulating an operating system (Rothenberg 2000, p.14) relies on users saving a complete detailed description of the operating system and its interaction with hardware, as well as a complete detailed description of the system architecture. This allows future users to build an emulator machine able to run old programs to interpret and display the archived documents.

D - Simulating Machine Behaviour

Legacy OS simulators developed to run legacy software require a level of system description that may prove elusive. This leads to machine simulators based on detailed descriptions of legacy machines in order to run legacy operating systems and legacy software. (Muir, 2001, p.168; Rothenberg, 2000, p.27) It is worthy of note that in the absence of museum machines for comparison purposes, machine simulators are best guess efforts. The question put forward by Bearman (1999) is whether preserving OS and / or machines is an attempt to preserve the wrong thing. Is the preservation target the system or the document?

2.3.5 Oltmans' E-Depot

Mentioned earlier in section 1.2 Oltmans (2004) advocated the e-depot approach to long-term digital document storage, built upon IBM's Digital Information Archiving System. Considering the ad-hoc nature of preservation strategies encountered thus far, the e-Depot is interesting in its systematic industrialised protocols. Unlike earlier preservation strategies, e-Depot advocates the preservation of the document, the tool
that was used to author the document, the operating environment where the tool was installed, and the hardware used. The hardware and the tools were assembled using many off the shelf component, when ingesting archive the system requires all relevant software installed on the Reference Work Station (RWS) and a snapshot image of the system is generated. When required the archived image is reloaded on an RWS for access (Oltmans et al., 2004 pp 279-284).

The dependence of the e-Depot on the hardware remaining compatible with the images generated at archive ingestion time “as long as the hardware remains the same” (Oltmans et al., 2004 p281), coupled with the complexity of the ongoing preservation processes required to maintain the viability of e-Depot such as:

- The need to identify from time to time publications in danger of becoming inaccessible due to technological changes
- Planning and implementing migration and emulation strategies
- Specification of both hardware and software required to render each document at ingestion time (Oltmans et al., 2004 p281)

Especially the specification of hardware and software in metadata renders the archival process fragile “One broken link is enough to make an electronic publication unusable” (Oltmans, 2004 p281). While in favour of this approach, is its ability to store in metadata the many alternate paths / tools for rendering some document types. In all the e-Depot remain hardware bound and dependant on proprietary software, operating system and a high level of regular user intervention to maintain its ongoing viability.
2.4 Preservation Trends

The past few years have seen the appearance of many archival algorithms and hypotheses; some are application specific to the organisations for which they were developed, and others are contenders for more generalised use. Publications encountered thus far advocate solutions based on proprietary systems, user compliance with guidelines at the time of document creation, record capture mechanisms, or restricted software catalogues. This more often than not stymies creative endeavours and produces adverse outcomes, as authors seek alternate avenues of expression. The ongoing work of Lorie, Waugh, Rothenberg, and others in the field of digital preservation is evidence that most of the approaches to date are not without flaws.

The fundamental aim is to explore a robust, low cost, low overhead, and widely available method for long term digital document preservation that is capable of transcending system and software application obsolescence while remaining capable of delivering to future users the digital documents stored as their original authors saw them.

2.4.1 Emulators, Simulators and Virtual Machines

Lorie (Lorie 2001, p349) developed a prototype to demonstrate emulation as a long-term strategy for digital preservation, which involves using a Universal Virtual Computer (UVC). The UVC approach is based on technical specifications for a simple, software-defined decoding machine being saved in paper format. He notes that the “machine” is not tied to specific hardware or software, and that the paper
copies (silverfish notwithstanding) could last for centuries. However, it remains to be seen whether such detailed technical specifications can be distilled into a brief paper document. It is also worthy of note that the demise of the paper would spell an unceremonious end to the digital archive.

Rothenberg (2000) experimented with and developed a system based on emulators, where original authoring software was preserved along with the digital documents together with a detailed human readable description of the architectural environment of the current machine, to be used by future users to build suitable emulators. The emulators would then emulate the original system functionality, thus permitting the original authoring tools to run and display the digital documents as seen by the authors. This approach, while premised on sound theory and experiments, nevertheless relies on thorough documentation of current system functionality, a task made increasingly difficult as current platforms evolve into ever-increasing complexity. The absence of comparative museum machines in the future will also become an impediment to test the integrity of newly written emulators. Franz (1993) asserts that the successful building of emulators for increasingly complex operating systems requires the developers to overcome major problems such as:

a) Incompatible paradigms, this may involve violating the rules of the target system to accommodate the new paradigm. Franz (1993) cites the example of the Oberon system, which is both a programming language and an operating system, originally developed as part of the NS32032-based Ceres workstation project; it is written entirely in the Oberon programming language. The entire system was designed and implemented by the Swiss Federal Institute of Technology Zurich (ETH Zurich, including Niklaus Wirth and associates).
The Oberon OS is also suited to several other hardware platforms, and is extremely compact. Complete with an Oberon compiler, assorted utilities including a Web browser, TCP/IP networking, and a GUI, the entire package fits on a single 3.5" floppy disk (ETH, 2000). Franz cites the Oberon system as being based on the paradigms of decentralised control and unlimited extensibility through the addition of modules that can be loaded dynamically, in contrast with the Macintosh operating system that is not freely extensible and is based on the application program paradigm, which distributes control to sub-parts that are statically linked to it. As extensibility was not anticipated by the designers of the Macintosh system, it cannot be built in it without breaking some of its rules (Franz, 1993, p678).

b) Instances of contrasting abstractions due to the divergence in the central abstraction upon which the different operating systems are founded, or where one or more concepts of the target system have no exact counterpart in the existing system, the designers are then forced to redistribute functionality. This may involve reconstruction of higher-level abstractions from lower level ones; Franz (1993) again cites the Oberon system differentiating between the abstractions of a data file and a mechanism to access it, in contrast with the Macintosh operating system that mixes the two concepts, associating a position with every file. Basing Oberon’s abstractions on the Macintosh is not possible without reprogramming some of their functions (Franz, 1993, p678).

c) Implementation restrictions that may be present in the target system which have no equivalent in the system being emulated could be, for example, limits on the number of concurrently open files (Franz, 1993, p679).
Waugh (2000, p175) alluded to the then current approach by the National Archives of Australia (NAA) namely "the post custodial archival model", where historical records are not put into an archive, but maintained in operational systems within an organisation. The goal was to reduce the cost of preserving the historical records. This was premised on two assumptions: first, it is not necessary to operate a separate archival system; second, since an organisation must migrate its operational records when upgrading the operational system, the marginal cost of migrating the historical records at the same time using the same software is negligible. It is, however, well worthy of note that since 2000 the NAA long term preservation policy is to take records selected for retention as 'national archives' into its custody and assume the principal responsibility for preservation and dissemination of these records (NAA 2000). Furthermore, Nelson (2001, p374) demonstrated that entrusting digital archives into the custody of a central repository authority is also fraught with danger. In the case of COSMIC (the official NASA software repository) which ceased operation in 1998, its responsibilities were turned over to NASA's technology transfer centres, and remains without a successor; this effectively means that the COSMIC archives of software are, for all intents and purposes lost at the time Nelson published his paper on Buckets (Nelson 2001, p374). Since 2001, the "Open Channel Software" with the consent of the National Technology Transfer Centre (NTTC) began an effort to locate and publish the COSMIC software collection created by NASA covering disciplines including engineering, chemistry, aerodynamics, and other areas. Open Channel Software does not maintain the software but rather offers it to the public under an "Adopt an Application" arrangement where many of the COSMIC programs are available for "adoption." When users adopt an orphaned application at Open Channel, they agree to moderate other users' contributions to the applications. They
also take over the maintenance of the site for the application through Open Channel Software Content Management system. The COSMIC collection consists of over 5000 programs. Open Channel Software thus far has an incomplete collection available online but undertakes to attempt to make available any program which has not yet been published. (Open Channel, 2004) The case remains that the software that helped land the lunar vehicle on the moon may no longer exist while the artefact (landing capsule) persists in a museum (the vehicle remains in serviceable order only it cannot be pressed into service as the software systems cannot be located!).

2.5 Latest preservation technologies

Among the leading preservation methods used are “Buckets” and the Victorian Electronic Record Strategy (VERS). These are discussed below.

2.5.1 Buckets

A “bucket” is an aggregative, intelligent construct for publishing. Buckets exist within the “Smart Object, Dumb Archive” (SODA) model, which can be summarised as promoting the importance and responsibility of individual information objects and reducing the role of traditional archives and database systems. (Nelson 2001) The goal is that smart objects will be independent of and more resilient to the transient nature of information systems. (Nelson 2001, p369) A bucket is a storage unit that contains data and metadata, as well as the methods for accessing both. The bucket design goals are: aggregation, intelligence, self-sufficiency, mobility, heterogeneity, and archive independence. Buckets are aggregative, intelligent, WWW-accessible
digital objects that are optimized for publishing in Digital Libraries (DL). As a
construct, a bucket contains the document as one of its elements and the application
which created the document as part of its metadata and a pointer to the location of the
application to view the document as another part of its metadata (Nelson 2001, p374).
The bucket approach reduces the knowledge acquisition overhead for future users,
rather than have future users discover which tools are suited to which document type,
the SODA construct contains linkages to the appropriate software to display the stored
digital document.

2.5.1.1 Why not buckets?

The tightly integrated nature of the bucket construct described earlier in 2.5.1 reduces
the tolerances to change. System changes, upgrades, and expansion have a
detrimental effect on buckets, especially if the application metadata becomes
outdated. Buckets contain pointers within their constructs to the application location
within the system, and the document location within the DL. System changes require
update to the pointers or the addition of a pointer at the original locations pointing to
the new locations. Over time the web of pointers becomes so entangled that the risk
of losses escalates to an unacceptable level. Failure to maintain the system often leads
to data loss, context loss, reference loss, and eventual archival documents loss.

2.5.2 VERS

VERS objects are based on encapsulation to wrap the information to be preserved
within metadata that describe aspects of this information. Metadata is provided for
both functional and organizational preservation. The metadata required for functional
preservation describes the data formats used, including the format of the encapsulation itself. The metadata required for organization preservation is more varied, but broadly describes what the preserved information is, its history, and its relationship to other preserved information. Organizational preservation metadata also covers authentication information that demonstrates that the information has not been modified since encapsulation (Waugh 2000, p179). The structure of a VERS Encapsulated Object (VEO) is also dependant on several conditions, namely:

(i) Self Documentation, as it cannot be assumed that future users will have access to the VERS documentation that will allow them to extract information from the VEO (Waugh 2000, p180),

(ii) Self sufficiency, in order to minimize dependencies on systems, data, or documentation (Waugh 2000, p180), and

(iii) Content Documentation for future users to clearly identify the data formats used to encode the content (Waugh 2000, p180).

Simply put, VERS is designed to be self documenting and self contained, where all the applications required to create, edit, and display a digital document are part of the VERS system and thus future users need not rely on outside documentation and tools to access the stored documents.

While VERS has many advantages it also has serious limitations. Where implemented, VERS constrains users to a set of applications software. It is thus suited for bureaucracies; creative work is often hampered by such constraints. The VERS process is still dependent on operating environment survival and proprietary software, and as such will not transcend system and software obsolescence issues. VERS is a complex system that captures the archival documents at the time of
creation from a specific set of authoring tools, and stores the documents encapsulated in metadata to be displayed by software tools that are also specific to the VERS system. While the system is maintained and updated to newer technology and platforms VERS will reliably present the documents as required. Should the system with all its complexity not be maintained and upgraded to suit new hardware, software, and operating platform, then, accessing the stored encapsulated digital documents will become a non-trivial task. By design the VERS system does not rely on external documentation, and when systems become obsolete, in periods as short as ten years, the documentation manuscripts are the only surviving useable artefacts of the system.

VERS was developed for a specific circumstance - namely the Victorian state government public sector. It has a limited market penetration and as such remains an untested system outside the government sector. The research indicated that VERS is a proprietary system and the source-code for the various components of VERS at this time remains unpublished.

2.6 Preservation trends

Partial reconstruction through fragments of physical archival material such as manuscripts, clay tablets etc is possible given today’s technology, but unlike early materials digital fragments are impossible to reconstruct. As already observed in Section 2.1, while it is possible to read the 400 year-old books printed by Gutenberg, unfortunately it is often difficult if not impossible, to read a 15 year-old computer disk (Kahle, 1997, p7). Long-term digital preservation efforts are hampered by the uncertainties of technological change. There is currently little experience on which to
base predictions of how frequently migration to new formats will be necessary or desirable or whether emulation will prove to be a viable alternative.

In the medium term, two operational approaches are noteworthy:

First, during creation, steps are taken that are likely to make migration or emulation less costly when they are needed.

Second, the bit streams generated by the migration process are kept alive through replication and routine refreshing supported by integrity checks (Arms, 2000).

Both these approaches do not address the issue of the archive surviving beyond the point where the external physical maintenance / support end. The two approaches are heavily reliant on user intervention to either physically migrate the files from one format to the next, and to replicate the migrated bit stream from time to time on fresh media. Furthermore, neither approach offers a way ahead for the archive to survive without the regular intervention of users. The question then arises: what becomes of the archive where no one is available to care for it? Will the archives based on such models become another “COSMIC” (Nelson, 2001, p374)?

2.7 Removable media and long-term storage?

The use of removable media in archival storage is widespread. Removable media like all other media has a finite life expectancy; media manufacturers specify the mean time between failures for each medium they produce. The problem when considering removable media is one of obsolescence in retrieval and playback technologies. As outlined previously computer hardware, storage, and software industries evolve at a rapid pace, yielding greater storage, and processing capacities at lower cost. Devices,
processes, and software for recording and storing information are regularly replaced with new, more advanced products and methods, often being driven by market forces. Records created in digital form are vulnerable to technological obsolescence. Even though some optical disk technologies have life spans of up to 100 years, it is arguable that for such media, longevity is of questionable value as current media outlast the software and devices needed to read them (Gilbert, 2003, p2).

Figure 2.4 Removable media expected life span (Gilbert, 2003, p2)

Figure 2.4 charts the expected life span of various media. It is worthy of note that these life spans are predicate on the availability of a viable retrieval / playback device (Gilbert, 2003). “At any given point in time, a particular format of a given medium can be expected to become obsolete within no more than 5 years.” (Rothenberg, 1999, p3)
<table>
<thead>
<tr>
<th>Medium</th>
<th>Practical physical lifetime avg.</th>
<th>Time until obsolete</th>
</tr>
</thead>
<tbody>
<tr>
<td>optical (CD)</td>
<td>5-59 years</td>
<td>5 years</td>
</tr>
<tr>
<td>digital tape</td>
<td>2-30 years</td>
<td>5 years</td>
</tr>
<tr>
<td>magnetic disk</td>
<td>5-10 years</td>
<td>5 years</td>
</tr>
</tbody>
</table>

Figure 2.5" the medium is a short-lived message" (Rothenberg, 1999, p3)  
Media often outlast retrieval / playback mechanisms, the issue of obsolescence is the principal hurdle when using removable media, as discussed later in Section 4.1.1; often users attempting to retrieve files from removable disks are confronted by the message illustrated in Figure 4.1.

### 2.8 Future Digital Archaeology

The literature research revealed that digital documents longevity is predicated on several conditions / circumstances. In concert, these conditions, the research indicates, would provide an optimal survival chance for a given archival document:

(i) Media longevity and technology-independence,

(ii) Refreshing bits, whether the physical medium is the same or not,

(iii) Migration of content from one data digital format to another, and

(iv) Emulation of the technical environment to function as if it were the technology of a previous generation to enable the use of obsolete software tools.

Today’s digital archaeology effort, such as Magnetic Force Microscopy (MFM) is already being used to recover data from media. While MFM is an extreme effort to
recover data (the worth of which may be unknown to future users), steps can be taken to reduce the effort required of future users to retrieve documents archived today. While the recovery of raw data from damaged media is made possible by MFM, interpreting the data stream into some intelligible message is not. Data sent for storage on any digital medium is manipulated and a variety of encoding and compression algorithms are applied as the data is written to the disk. The recovery of the original message is hence dependant on users employing suitable cryptographic techniques interpret the raw data stream and break it into its original meaningful units” (Ross, 1999, p32-34). In the absence of these cryptographic techniques (due to device / software / driver obsolescence), the recovery effort required of future users is a non-trivial effort especially for data the value and quality of which is unknown.

Future Digital Archaeology (FDA) is concerned with facilitating the recovery effort where preservation fails or is likely to fail. FDA is the contingency of last resort, if all else fails the provision of sufficient data assists in figuring out how to read stored bits and work out what they mean when other archiving methods fail. In an effort not too dissimilar to the Rosetta-Stone, FDA delivers to future generations the data, the applications to view the data, and the science behind the applications to view the data. In short, today’s digital archive may still be available to future users for a minor expenditure of time and effort.

2.9 Conclusion

‘An Open Source Experiment’

A common theme permeates the literature dealing with archival longevity. This theme revolves around the development of a single unique solution that would become a panacea to all digital documents under all circumstances. Thus far, such a
solution has proved elusive. Over the past century technologies have evolved at a rapid rate, no sooner than inventions became innovations they were superseded by newer innovations, the world has gone to war twice empires have fallen. The successful survival of digital artefacts should be independent from user intervention; their can be no guarantee of a guardian that will diligently replicate digital documents as technology evolves, or migrate documents as application become obsolete. A viable preservation process must take into consideration survivability beyond the constant intervention of the users. The preservation efforts outlined in this chapter are all reliant on some measure of periodic user intervention, or high degree of prior contents knowledge in the case of genomic storage.

Based on this information, an opportunity has been identified for an innovative approach to long term digital archive survival utilising software developed under the GNU General Public License system (GPL), where recipients have all the rights to copy, store, distribute, use, and/or modify the source code. Access to the application’s source code would give future users insight into the workings of the applications. The meaning of the stored digital documents (if the format is other than simple text) may then be discerned, or better still, the applications can be modified to run within future operating environments.

Digital documents need means to survive the disappearance of tools and systems that were used to author them. While the focus of the published research thus far is on using a mix of proprietary software legacy operating systems and regular user intervention, this research seeks to use non-proprietary software and operating systems to circumvent the disappearance of authoring and rendering tools as well as adapt to the continuously evolving hardware technology.
Chapter 3 - RESEARCH METHODOLOGY

This experiment examined the long-term survival of digital archival documents to well beyond the demise of the authoring and viewing technologies employed to create them. The experiment also considered the question: were the systems and platforms used to run these technologies favourably aided by the co-archiving of compatible Open Source tools and build / deploy instructions? To describe the research design, this chapter provides an outline of the experiment, and provides details of the local area network infrastructure where the experiment was conducted. This chapter describes the Open Source applications used, provides program descriptions for each application, and evaluates the compatibility of Source of Open Source software tools.

The experimental method describes the use of the Open Source suite of tools to render, edit and save files from the experimental archived collection of documents. It further evaluates their usability and compatibility with proprietary file formats such as Microsoft (documents, spreadsheets, presentations etc...). This chapter also addresses disk usage and storage and media issues, in addressing disk usage it considers compressed archives as a redundant backup mechanism. Finally, this chapter investigates document retrieval, and describes experimental data collection and treatment.

The above will support the premise that Open source tools and applications are a viable additional measure to ensure archival accessibility in the medium to long term.

This chapter provides a detailed overview of the research experiment; the aim of this research was to provide one possible solution to long term archiving using “Open Source” software tools and currently available applications. This circumvents many of the problems encountered when using proprietary software on platforms and
Operating Systems for which they were not intended. The research will draw on some existing methods and will propose others, but will be based on Open Source. The reason for using Open Source as opposed to proprietary software is the ability to store the application and operating system source code along with the archived material, as proprietary applications source code is seldom published. The rationale for storing the source code is the Future Digital Archaeology (FDA) premise of being the contingency of last resort discussed in Section 2.7 namely where future users have access to the software source code with the aim of modifying it to render it compatible to their contemporary systems, and then use it to access the archived digital documents created by proprietary software. Future users will then be able to see documents from the archive as their original authors saw them at the time of creation.

The experiment archives a collection of digital documents on a variety of media, a redundant copy of the collection being stored in compressed form on a commensurate assortment of media. Digital documents stored on any media for an extended period will be subject to degradation such as noise, which may be brought on by poor storage condition or as part of the natural process of decay, (the law of Entropy states that order will decay to disorder over time). Magnetic disks over time will become unreadable. Likewise, CDs suffer from Oxidisation and Fungus problems as discussed in Section 2.2.2.

That part of the archive residing on hard disk is subjected to artificially introduced noise to simulate the passage of time. Unlike naturally occurring noise, the artificially introduced noise was distributed (the nature and distribution is described in 4.3). It is also worthy of note that naturally occurring noise may or may not be distributed. In an attempt to mimic naturally occurring noise from the passage of time the
experimental process was repeated over 4 cycles, with the amount of noise introduced being increased at each iteration. The initial cycle started with an arbitrary amount of 3% noise, then 7%, then 11%, and finally 18%. The nature of the artificially introduced noise (in other words its distribution) is discussed in detail in Section 4.3 of chapter four, which contains the analysis of the experimental results of this thesis.

### 3.1 The Experiment

This experiment attempted to simulate conditions enabling retrieval using Future Digital Archaeology (FDA). The rationale is that when other retrieval efforts fail FDA will provide sufficient data to assist in reading the stored bit-stream, and correctly interpret the archived data. FDA will deliver to future users the data, the applications to view the data, and the science behind the applications to view the data. In short, today’s digital archive will still be available to future users for a minor expenditure of time and effort. The experiment explored storing an archival collection and a commensurate collection of software tools together with the source code for the software tools. The experiment rationale for storing the source code is that as platforms and operating environments change and evolve over the years, it is unlikely that the source code will compile cleanly in new environments. However, since source code is always simple ASCII text, future users may be able to modify / improve it to the point where it compiles to an application able to run on their system and thus have a viewer / editor able to display / render archival files as their author saw them at the time of creation. In the case where the language (code) has fallen into disuse, as computer languages have published standards, future users may refer to these published standards and develop compilers suited to their computing platforms.
to compile the archived code. Admittedly this is a non-trivial exercise, however it is worth noting that this will be a once only exercise, as once developed the compilers may then be used repeatedly.

3.1.1 Experiment outline

The experiment comprised three stages: The first step was the collection of a pool of Open Source software tools source code, test compiling, and deployment of executable applications. The second step was collating an archive comprising a number of file format documents created using modern proprietary software tools. The third step was storing the archive along with the Open Source application source code on various media including network nodes utilising unused hard disk space, and simulating the passage of time by introducing noise, as described later in section 4.1.3 and 4.1.4, to the collection (only possible on multiple read/write media). CDRom as opposed to CDRs and magnetic disk are read only media and as such once created (pressed or written and finalised) may not be altered, the only possible artificial noise introduction is by physically damaging the media by mechanical means (Kozierok, 2001).

3.2 The Network

The network nodes used for this experiment were all part of a single cluster within the School of Information Technology and Computer Science (SITACS). The cluster comprised 36 Sun Ultras interconnected by 10/100 copper twisted pair Ethernet to a Cisco Catalyst 3500 XL 48 port switch, then by 1 gigabit fibre optic to the building.
switch, thence to the University of Wollongong server; Figure 3.1 illustrates the experiment cluster and its connection to the University network. The 36 Ultras comprising the experimental cluster each boasted 30 gigabytes of unused disk capacity, since network machines account holders on these machines had storage space allocated by NFS on remote RAID devices. As a result of this arrangement the available storage with this one cluster was 1080 Gigabytes (1.08 Terabyte).

![Network diagram](image)

Figure 3.1 Network diagram (Java Lab at SITACS within the University of Wollongong)

### 3.3 The Open Source Applications

The representative Open Source applications were drawn from the large pool of source archives available from Source Forge. Source Forge is a repository of Open
Source code and applications free to Open Source users and developers, Source Forge files are freely accessible via the internet and require only a network connection no payments, no memberships (http://sourceforge.net/, Source Forge, 2003). The rationale for using Open Source tools is that it is possible for users to view and edit the application source code with the view of porting it to their systems; it is also possible to develop system specific compilers to build system compatible applications.

3.4 How can Open Source be used?

While the aim of the Open Source project is to allow contributors to revise and otherwise improve the applications, this aim does not preclude the static preservation of the source for later use or adaptation. The rationale for storing the code of Open Source applications was to provide future users with a contingency of last resort the complete code for a viewer based on the technology of the archival document era ready for compilation, if they have no compatible viewer to use.

3.5 Advantages of Open Source Software

The availability of the source code and the right to modify makes it possible to port the code to new hardware with the view of adapting it to the changing architecture and operating environments.

The extendibility of an application's lifetime is always dependant on the availability of its source code. The fact is that no binary-only application survives unmodified for more than 10 years, however research has identified Open Source software systems
from the 1980s that are still in widespread use (adapted to new environments such 
application include but not limited to XPDF and Emacs). It is worthy of mention that 
under the GPL licensing systems the ported source code must be published as Open 
Source also (González-Barahona, 2001, Sec 4.3).

The ability to store, copy, and install the applications at will without a licensing fee as 
Open Source licence arrangements grants rights for the free redistribution of modified 
and improved applications' code as well as the right to reuse any Open Source code in 
the development of new applications.

Unlike proprietary software there are no vendors to restrict in any way how the 
software is used, and where the common concern with proprietary is the manufacturer 
closing down, or deciding to discontinue development of the product. For Open 
Source software this concern is moot; if the group or entity that originated the code 
decides to stop development, it is always possible for another to continue the 
maintenance, and improvement, without legal or practical limitations (González-
Barahona, 2001, Sec 4.3).

No "black boxes" are possible. This point is so important that Open Source is now 
considered by experts as a necessary condition for dependable applications. The 
reason for this importance is dependability of the services provided by the given 
software. In other words by having the source code available, it is possible to perform 
a thorough inspection and verify the correctness of the algorithm and the 
implementation scheme used (González-Barahona, 2001, Sec 4.3).

There are fewer conflicting priorities due to marketing pressures. This is a simple 
consequence of the fact that there is no single commercial entity pushing for precise 
delivery dates or features that must be supported. Usually Open Source software is
delivered "when it is ready", and when the development team feels that its quality is good enough. This means that software usually does not need as many "service packs", updates and such. Service packs and patches become available when vulnerabilities are discovered and someone writes and tests them (González-Barahona, 2001, Sec 4.3).

3.6 Could proprietary software perform the same functionality as Open Source?

The question as to whether proprietary software fulfils the task Open Source software performs in this research; the simple answer is “no”. The reasons are:

- Proprietary software source code is seldom published.
- License / patent owners of the software provide a license to use only.
- The license obtained by purchasing the software precludes copying the software; it permits only one installation.
- Archiving a collection for extended periods does not mean this license condition will not be breached in the future.
- The license does not permit users to modify the software which is sold as binary executables only and software licences prohibit reverse engineering (executable files cannot be modified; in order to modify an executable program, the machine readable source code must be modified).
3.7 Why Open Source?

The reasons for using Open Source software and operating system in the context of this research are:

- Open Source is free; it is fee free, but more importantly, the users are free to alter the software to improve it. Altering the software to make it run on platforms it was not written for (porting) is improving the software (González-Barahona, 2001, sec 4.3).

- Open Source is publicly available in source code form as well as installation packages. In reality some installation packages are tar-balled (tar is part of a compression process widely used by Unix/Linux users) source files with a script to decompress build and deploy the built applications.

- Open Source applications are available to view, edit and save files created by today’s proprietary software. So while proprietary software source code is unavailable to future users to modify and adapt to their computing platform in order to view the archived documents, compatible Open Source software tools are stored alongside the archive with the expressed intent that the code be available for use.

3.8 Advantages of Open Source operating system

The source code for Open Source operating systems is also freely published and available for modification/improvement; storing the Operating System source along with the archive provides future users with valuable insight into current technology. In order to successfully develop compilers to suit the stored software tool’s source
code, the operating system source code becomes a valuable resource to developers. It provides insights as to how current software addresses the hardware peripherals, how it accesses devices and memory, as well as how the current machine executed instructions, responded to interrupts and so on.

3.9 The experimental method

The experimental method involved developing a simple single table database of all file types archived, proprietary authoring software used to create them, along with corresponding Open Source tools and their build instructions files. This table was stored in ASCII text format on each medium used. The table will inform users of the file type and the appropriate Open Source software tool to view the document.

The experiment collected a pool of Open Source software source code, then compiled and installed it within an Open Source operating system. The experiment then tested the software tools appropriateness using data files free from noise / alterations created by proprietary authoring applications.

3.9.1 The First Experimental Cycle

This first iteration established a benchmark of suitability in other words the first cycle served to demonstrate the appropriateness of software tools to the digital documents archived. Each application was used to open appropriate files from the archive, the opened documents were examined to determine correct display of contents, whether the files had functionality such as macros or embedded formulas, and so on. The functionality of these was tested, and testing logs were kept.

S. M. Antoun

Chapter 3
Following the first cycle the experiment was repeated with data files containing introduced noise as described in Section 4.1.3-4 in an attempt to simulate naturally occurring noise from the passage of time. The experimental process was repeated over 4 cycles and the amount of noise introduced was increased at each iteration, the initial cycle started with the arbitrary amount of 3% noise, then 7%, then 11%, and finally 18% for the purpose of determining the degree of resilience / fault tolerance of each file format.

3.9.2 Program Descriptions

This section describes the various programs used in the experiment; Open Source software tools are described in detail and comparatively evaluated against proprietary programs. The aim was to resolve the following issues:

- Compatibility of Open Source tools with data files authored by proprietary software

- The viability of using unused storage space on network nodes for long term archival purposes

- The effect of the passage of time on the archive as simulated by artificially introduce noise

- The viability of compressed archives as a redundant backup mechanism.
3.9.3 Compatibility and Source of Open Source Software tools

All of the programs utilized were Open Source software freely distributed under the GNU GPL licensing system or developed specifically for this experiment using the GNU development platform and as such are also available as Open Source. Additionally the operating system used for this research was also an Open Source operating system, namely Linux Mandrake (public release November 2002).

In order to ascertain the viability of Open Source tools for the purpose of the experiment, a representative sample of tools compatible with most popular proprietary software in use today was chosen. The sample covered desktop publishing, graphical editing, video playback and other audio visual tools. The collection of software tools used to test compatibility of Open Source software with files authored with proprietary software comprised the following:

- "Open Office" Desktop publishing tools
- "Xpdf" PDF reader
- "Xine" Video/DVD player
- "The Gimp" Graphics
- "XMMS" CD player.

The tools source was downloaded from: http://sourceforge.net/ (Source Forge, 2003), "SourceForge" as described earlier is a repository of software source code available to the Open Source community. The software was built according to the accompanying instructions (a non exhaustive set is included in Appendix - "A"),
and deployed within an Open Source operating system. The aforementioned tool was then used to open, edit and playback data files produced by proprietary software. As part of the experimental process OpenOffice and its competitor Microsoft Office XP were comparatively assessed to establish a baseline for the assessment of digital document files cross applications compatibility measurements.

### 3.9.4 Assessment

OpenOffice and Microsoft Office XP were comparatively assessed to ensure the compatibility of OpenOffice with digital documents created using Microsoft office tools. This ensures that suitable viewers are stored in source code along with the archive in line with the research rationale discussed in chapters 1 and 2.

### 3.9.5 Suite of tools

OpenOffice includes most of MS Office’s core features, spelling checker, auto-sum, graphing compatibilities in the spreadsheet, and wizards for assembling slide shows.

### 3.9.6 Usability

Switching task between Writer,Calc, Draw, and Impress documents in OpenOffice via a menu of all open files is a distinct advantage for users. The user interface of OpenOffice is somewhat similar to MS Office, and is highly customisable.
Users may with little effort bring the Graphical User Interface (GUI) to one that is almost identical to Office XP or Office 2000. All but the most obscure keyboard shortcuts are the same in both suites.
3.9.7 Compatibility with Microsoft file formats

OpenOffice had mixed results. MS Word, MS Excel, and MS PowerPoint documents could be opened, viewed and manipulated with most of their formatting and functions intact, but MS Word macros were incompatible with Open Office. OpenOffice also exports to MS Office formats; the file conversion from OpenOffice “swx” format to Microsoft “doc” is error free.

3.9.8 Graphics handling

The research briefly evaluated the graphics rendering / editing tools available within each office suite. The research considered the quality, clarity, and overall presentation of the graphics, this comparison was viewed using an LCD monitor on an LS400 Dell Latitude laptop. MS Office graphics rendering is superior in quality and resolution to Open Office. The OpenOffice graphics tool “Draw” creates flat-looking images; the OpenOffice spreadsheet tool “Calc” provides only 2D charts, while 3D bar charts are available in MS Excel.

3.9.9 Application help tool

The research briefly evaluated the help tools available within each office suite. The research considered the quality, clarity, context and presentation of the help files. Open Office’s online help is incomplete, terse, and lacks context; by contrast, Office XP’s help online has step-by-step instructions.
3.9.10 Integration

The research briefly evaluated the integration of the suite of tools available within each office suite, the research considered the quality, and overall seamlessness of the integration. OpenOffice does not offer an e-mail or a calendar application, while MS Office has Outlook to handle necessary e-mail tasks, as well as a well-integrated calendar. Furthermore, MS Office offers collaborative editing tools, while OpenOffice fails to support collaborative work.

3.9.11 Disk usage

The research briefly evaluated the disk usage of each office suite. The research considered a complete install of both programs, OpenOffice used 233 MB fully deployed, as opposed to Office XP’s 275 MB of required disk space for a typical install.

3.9.12 Operating Systems supported

The research briefly evaluated the operating system supported by each office suite. The research considered the quality, clarity, and overall presentation of the graphics. MS Office XP runs on Win98, 98SE, 2000, NT4.0 with SP6, and XP (Home and Professional). OpenOffice runs on Win98, 98SE, 2000, NT, XP, Solaris 7-8, Linux Kernel 2.2.13 or higher & glibc2 V2.1.3 or higher (Turvey, 2003, p2). Table 3.1 presents a comparative diagram of the features of OpenOffice and MS Office, aside from minor aesthetic inferiorities OpenOffice is compatible and commensurate suite of applications to MS Office.
<table>
<thead>
<tr>
<th></th>
<th><strong>Open Office</strong></th>
<th><strong>Microsoft Office</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format Compatibility</strong></td>
<td>Supports “swx” “doc” “rtf” file formats</td>
<td>Supports “doc” “rtf” file formats only</td>
</tr>
<tr>
<td><strong>Graphics handling</strong></td>
<td>Lower quality graphics, able to display MS office graphics but lesser quality and no 3D graphics</td>
<td>High quality, crisp sharp well textured graphics and 3D graphics</td>
</tr>
<tr>
<td><strong>Help files</strong></td>
<td>Terse, incomplete, no context and no step-by-step instructions</td>
<td>Complete, good context and step-by-step instructions</td>
</tr>
<tr>
<td><strong>Integration</strong></td>
<td>Applications exist within the suite of tools but are poorly integrated; often require manual start up no collaboration support</td>
<td>Seamless integration e-mail &amp; calendar; allows for collaborative work.</td>
</tr>
<tr>
<td><strong>Disk Usage</strong></td>
<td>275 MB of space full install</td>
<td>233 MB of space full install</td>
</tr>
<tr>
<td><strong>OS supported</strong></td>
<td>Win98, 98SE, 2000, NT, XP, Solaris 7-8, Linux Kernel 2.2.13 or higher &amp; glibc2 V2.1.3 or higher</td>
<td>Win98, 98SE, 2000, NT4.0 with SP6 only.</td>
</tr>
</tbody>
</table>

*Table 3.1 Comparison of Office suites...*

The applications selected for the experiment as outlined provide similar functionality and usability to those most widely used proprietary applications, and as such will access a vast majority of archival documents produced by today's users.

### 3.10 Storage and Media

The collection was stored on an assortment of media: magnetic fixed disks, optical media, and magnetic removable disk (floppy disk), magnetic removable mass storage disks (zip disk), in order to study the effects of the passage of time and noise (introduced or otherwise) on the archival collection.

The part of the collection residing on hard disk was sent to storage on remote sites utilising "Archiver" - a multi-threaded Parallel Virtual Machine (PVM) program.
(code listing is included in Appendix B) developed for the experiment. The multithreaded parallel design was used to support management, transport, distributed access to the data in order to optimise performance when transporting large amounts of data over the network. This design is also utilised in climate modelling and other fields where large data sets are accessed over networks (Allcock, 2001, p3). A fragmented redundant compressed copy of the archive created using the “RAR” compression tool (RARlab 2003) was also stored on the network nodes.

The storage space differed from a normal user’s account space in a UNIX environment as follows; users in a UNIX environment are allocated space on Network File Server (NFS). This space may physically reside on one or more storage devices. Access to these devices is often transparent to the users. A user may simply write to /home/user-id/... . For the purpose of this experiment space was allocated on each network node’s local drive manually outside NFS, and as such the local space on each drive was accessible only when logged on to the individual machine using /spare0/user-id/... . The advantage of this is that when a process such as a PVM daemon is spawned to run a process on a remote node, this daemon will run as the owner’s User Identifier (UID), and as such will be able to access the owner’s account space on the local drive. This arrangement enabled spawned PVM processes to write to the local hard drive on each remote machine. Spawned processes run as owner UID were able to access both /tmp (public directory) on the local hard drive or specified location within user allocated space (/spare0/user-id). The above also meant that files written to the local drives on network nodes were owned by the user, and as such only accessible to the account owner and super user. This setup meant only users with allocated space on individual network nodes have access to this storage method. It did however demonstrate the ability to extend this process to beyond the
bounds of local network nodes; it demonstrated that storage can occur wherever a user has allocated storage space and is able to run a process. The rationale for using a multi-threaded PVM tool is that using multiple transport threads should result in far better overall performance than using a single thread over the network to transport vast amounts of data.

A complete copy of the archive was also stored on 3 optical disks (TDK CD-R Gold 700 Mb/80 min). A subsection of material from the archival collection was stored on magnetic disk (Iomega zip100 disk). Finally an even smaller selection of documents was stored on 3 standard 3.5 inch 1.44MB floppy disks.

3.11 The viability of compressed archives as a redundant backup mechanism.

A redundant copy of the archival collection was stored as a compressed archive using the RAR compression tool. While RAR compressor is not Open Source software, the decompressor is, and as such falls within the experimental scope. The rationale is that future users will only need a decompressor to recover the archives using the redundant compressed volumes.

3.12 Context retention

The experiment maintained a central catalogue of archived files and the network node where these reside. As a further measure to ensure robustness, redundant copies of the catalogue were stored on all network nodes as part of the archived collection. The catalogue is in simple ASCII text and requires a basic text editor on any platform to
open and read. Whenever the archive was modified either by adding or removing files the central catalogue was updated then copied to all network nodes overwriting the previous catalogue.

3.13 Retrieval

The archive retrieval was designed to be documentation based. The archival catalogues detailing the collection, and the whereabouts of files as well as their appropriate viewing tools were available uncompressed for users to view using simple ASCII text viewers (where users may wish to view the entire recreated collection, this would involve the staging (gathering) of the archive from the various storage media and sites on disk cache). An appropriate Open Source software viewing tool then accessed the desired documents. In order to test the recovery process based on the redundant compressed volumes the source code for the RAR decompression tool was compiled. The decompressor was utilised to recreate the archive from the redundant compressed volumes. The decompression tool utilised its error recovery functions to recreate removed archive fragments. The restored archive was then ready for use.

The redundant compressed archive provided the only error correction / recovery in this experiment. In any archiving effort, error recovery mechanisms are an essential tool in maintaining the archive’s viability. Using multiple redundant copies may also provide a recovery mechanism where future users mix and match fragments from the multiple copies in order to rebuild an error free archive. The experimental process has demonstrated the need for error recovery measures in long term archival efforts.
3.14 Data collection and treatment of the data.

In this section The research established a set of measures for evaluating the retrieved archive and the ability of the suite of tools to display documents from these archives after the introduction of artificial noise described in Sections 4.1.3 and 4.1.4 to mimic the passage of time. A set of four criteria were determined as viable indicators to be utilised to determine both:

1) The cross platform and cross application compatibility of the experiment’s tools, and

2) The viability of the files retrieved from the archive after the introduction of noise.

These measures are:

- Lossines of conversion (whenever a conversion was necessary),
- Edit-ability of the data files,
- Ability to save back to the original format, and
- Functionality retention (embedded formulae, macros, hyperlinks).

A Boolean measure was gathered for each applicable criterion. In other words, each file was opened using an appropriate Open Source software tool. If the file successfully opened with no error messages then a score / measure of “true” for “no-lossiness” was recorded. If the software converted from the file’s original format to the format it requires (in the case of the spreadsheet from coma separated values) and displayed correctly, a measure of “true” for “no-lossiness” was recorded. If a file
converted from one format to another with some losses (rich text format to simple text), a measure of "false" for "no-lossiness" was recorded.

Similarly this measuring process was applied to Edit-ability, Ability to save, and functionality retention. In the case of edit-ability if the application opened the file as read only it was recorded as a "false" (with the exception of PDF files). In the case of ability to save, a warning message about some formatting loss for saving to a format non native to the Open Source software was ignored. Only a fail to save was recorded as a "false".

As to functionality retention the loss of macro functions or the loss of functionality of embedded formulas was recorded as "false", the failure to playback in the case of video / audio was also deemed "false". In order to validate the premise of the experiment, a set of measures of the four criteria was taken after the disbursement of the archives over the network nodes, and again after individual iterations of the experiment cycle as outlined in Section 3.9.1. The results of these measures are detailed in Chapter Four along with a discussion of these results.

The sources for data files used in the experiment are:

1. The author's own collection of documents accumulated over six years of studies,

2. Files obtained from various sources over the internet - mainly electronic libraries where permission is given to copy a digital or a hard copy of the work for personal / educational use,

3. Files available under public license, and

4. Promotional material publicly available over the internet.
3.15 Conclusion

This experiment sets out to test the aim of this thesis to explore a robust, low cost, low overhead, and widely available method for long term digital document preservation that is capable of transcending system and software application obsolescence, while remaining capable of delivering to future users the digital documents stored as their original authors saw them. The methodology outlined aims to demonstrate that Open Source application source code stored along with the archives provides a means for long term digital archival success and that this method is a simple method that requires minimum interventions / refreshing / migration. To that end the experiment involved four iterations; on each cycle the amount of artificially introduced noise, simulating the passage of time and deterioration of storage media, was increased. The archive collection was then inspected and data as to the ability to retrieve and display according to the criteria outlined in Section 3.14 was gathered. Analysis of this data is the subject matter of Chapter 4.

The research anticipated that along with the increasing amount of artificially introduced noise the number of files that fail to open and display correctly increased. The anticipated result is based on the simple but true fact: If a file is subjected to increasing damage it would eventually reach a stage where it ceases to be usable.
Chapter 4 – RESULTS

4.1 Background to the Experiment

As outlined in earlier chapters this research set out to investigate the possibility of using Open Source software to promote the long term survival of digital documents. The research also looked at the viability of utilising unused data storage capacity on network nodes for long term archival purposes, in order to construct a long term archival model that is both reliable and free from constraints normally found within proprietary systems.

The first research question was: How do files authored by proprietary software survive the obsolescence of the viewing software? As outlined in Chapter 3, the reason for using Open Source as opposed to proprietary software is the ability to store the application and operating system source code along with the archived material. The rationale for storing the source code is the FDA’s contingency of last resort discussed in Chapter 2, where future users have access to the software source code with the aim of modifying it to render it compatible with their contemporary systems, and then use it to access the archived digital documents created by proprietary software. Future users will then be able to see documents from the archive as their original authors saw them at the time of creation. The advantages of Open Source over proprietary software in long term archiving as previously discussed in Chapter 3 are:

- The ability to view and edit the application source code (Wang, 2001, p96) (Open Source Initiative, 2004).
• The ability to port the application from one platform to another (it is worthy of mention that under the General Public Licence (GPL) licensing systems the ported source code must be also published as Open Source (Open Source Initiative, 2004).

• The ability to modify the software functionality to suit users’ requirement (Wang, 2001, p96) (it is worthy of mention that under the GPL licensing systems the source code may be modified and may or may not be published as Open Source but may not be rendered proprietary) (Open Source Initiative, 2004).

• The ability to store copy and install the applications at will without a licensing fee (Wang, 2001, p97) (Open Source Initiative, 2004).

The flexibility afforded by the GPL licensing system coupled with the availability of the application source code renders Open Source software uniquely suited for the purpose of long term archival survival. It is the ability to edit/modify the application’s source that will give it the longevity needed to promote the survival of the archive beyond the current era of technology.

4.1.1 Open Source Software

This research involved searching the Open Source community for suitable software tools to view, edit, and save files authored by proprietary software. This research question also led to the comparative analysis between proprietary software and Open Source software in the most common area of modern computing desktop publishing (discussed in detail in Chapter 3).
A related research question was: Will the archived data survive the test of time? The real test required for answering this question is a longitudinal study spanning several decades. It is however possible to mimic the effect of the passage of time by introducing noise / distortion artificially to a collection of digital documents. The rationale for this assertion is the observed effect of time passage on media and its contents. Anecdotal evidence from attempting to access data stored on old magnetic media revealed distortions, with deterioration often leading to a total failure of the media. Often users attempting to access a file on magnetic disk that has not been used for a number of years results in the familiar message “Disk is not formatted”

![Disk is not formatted](image)

*Figure 4.1 Error encountered when attempting to use a 3.5 inch magnetic disk unused for a few years.*

The data forming the basis of the following discussion are the collected results of experimental simulation of the passage of time - namely the effects of introduced noise on the digital archives.

### 4.2 File types

The experimental process involved testing a selection of files generated by proprietary software tools, these were:
1- MS Word files "*.doc" comprised several subcategories depending on contents:

i. Documents with embedded Cyrillic characters,

ii. Documents with embedded graphics,

iii. Documents with embedded images,

iv. Documents with embedded hyperlinks,

v. Documents with embedded tables,

vi. And documents comprising text only.

2- Spreadsheets with formulas "*.csv"

3- Portable Document Format "*.PDF" generated by a variety of applications

4- MPEG4 video files "*.mpg"

5- QuickTime movie files "*.mov"

6- Jpeg image files "*.jpg"

7- Audio files

8- Program source code (a sample of the Open Source applications source code files)

4.2.1 MS Word documents

Widely used software application MSWord documents account for a substantial proportion of any given document collection. Files created using this application vary
in size content and characteristics; for the purpose of this experiment the following
type of .doc files were used:

A- “.doc” files with embedded Cyrillic characters. These files are typically bilingual
files with sections edited in English and other sections edited in an eastern European
language. These files are typically fragile as they contain multiple fonts and
encoding. When damaged the secondary encoding is lost and corresponding
characters are substituted with “?”. The primary encoding is dependent on the
installation, where the operating system and the application are English by default
Cyrillic becomes the secondary encoded language in the .doc document (and requires
some additional files on first use (downloaded and installed from Microsoft website
(www.microsoft.com)). Similarly, if Cyrillic is the default language of the operating
system and software application English becomes the secondary encoded language
within the “.doc” document - (and requires some additional files on first use
(downloaded and installed from Microsoft website (www.microsoft.com)).

B- “.doc” files with embedded graphics. These documents contain a combination of
edited text and graphics, the graphics being either imported into the documents using
the insert menu item or generated using the in built functionality of the application.
The imported graphics can be in a variety of formats from Windows Meta File (.wmf)
to Graphic Interchange Format (.gif). The graphics are always saved within the
documents and are not required as a separate file for the document to display correctly
(unlike graphics within an html file).

C- “.doc” with embedded images. These documents contain a combination of edited
text and images, the images being imported into the documents using the insert menu
item. The imported images can be in a variety of formats, from Windows bitmap File
MSWord is capable of interpreting 21 of the 64 currently available image file formats. The images are always saved within the documents and are not required as separate files for the document to display correctly (unlike graphics within an html file).

D- “.doc” with embedded hyperlinks. These documents contain a combination of edited text and hyperlinks, the latter being embedded into the documents using the insert menu item. They can point to other documents, locations within a document, an html page on the World Wide Web, an html page on the intranet, newsgroup, gopher, telnet, or an FTP site.

E- “.doc” with embedded tables. These documents contain a combination of edited text and one or more tables; tables are inserted into the documents using the table menu item. They contain data in tabular format such as the data presented in Figures 4.2 to 4.5 inclusively.

F- “.doc” comprising text only. These are the simplest form of document comprising ASCII text and other non printable formatting characters.

Microsoft’s office productivity software “Office”, accounts for 90 per cent of office productivity software sales. It follows that the majority of word processing documents produced today are “.doc” (MS Office Word (word processing format)) files; as such the experiment used a large sample of word processed files to reflect this distribution. The multiple types of documents described from Sections 4.3.1.1 A through to F reflect the wide variety of documents types produced by MS Word, and aim to provide a representative sample of the various possible archival “.doc” documents.
4.2.2 Spreadsheets with formulas

A spreadsheet is a table of values arranged in rows and columns. Each value can have a predefined relationship to the other values. If a change occurs in one value, a commensurate change may be necessary to other values as well. Spreadsheet applications (sometimes referred to simply as spreadsheets) are computer programs that permit the creation and manipulation of spreadsheets electronically. In a spreadsheet application, each value sits in a cell. Users can define what type of data is in each cell and how different cells depend on one another. The relationships between cells are called formulas, and the names of the cells are called labels. The relationship between the cells is the feature this experiment alluded to as functionality retention.

4.2.3 Print Document Format “.PDF”

PDF is a particular file format, like .doc or .xls files. PDF is built on the PostScript language; postscript files are “programs”, actual application created, sent and then run inside the PostScript RIP (Raster Image Processor). PostScript is designed to describe a page. PDF does that as well, but beyond this, PDF can also contain information not only related to how a page looks, but also can describe how it behaves and what kind of information is contained in the file. A PDF file can contain fonts, images, printing instructions, keywords for searching and indexing, job tickets, interactive hyperlinks, movies, and so on.
4.2.4 Jpeg files “.jpg”

The Joint Photographic Experts Group (JPEG) is a standardized image compression mechanism designed for compressing full-colour or grey-scale images of natural, real-world scenes. It works well on photographs, naturalistic artwork, and similar material; it is less suited for lettering, simple cartoons, or line drawings. JPEG handles only still images, there is a related standard called MPEG for motion pictures. JPEG is “lossy,” meaning that the decompressed image isn’t quite the same as the original. JPEG is designed to exploit known limitations of the human eye, notably the fact that small colour changes are perceived less accurately than small changes in brightness. Thus, JPEG is intended for compressing images that will be viewed by humans. Jpeg compression is widely used on websites where the trade-off is between acceptable image size and quality; web designer often choose to trade-off some margin of image quality in return for smaller files that transmit quickly.

4.2.5 QuickTime movie files “.mov”

QuickTime is a format for the creation and distribution of video, audio and other formats. Now an ISO standard, QuickTime was initially developed by Apple Inc. for use in Macintosh computers. Although it comes with the System software distribution, it is updated periodically by Apple. QuickTime is encoded by a codec (enCOder/DECoder) such as Sorenson in a Macintosh environment or Intel’s Indeo in a PC environment.
4.2.6 MPEG4 video files “.mpg”

The Moving Picture Experts Group (MPEG), “mpg” format is based on a family of standards used for coding audio-visual information (such as movies, video, music) in a digital compressed format. The major advantage of MPEG compared to other video and audio coding formats is that MPEG files are much smaller for the same quality. This is because MPEG uses very sophisticated compression techniques.

4.2.7 Audio Files

Audio File Formats describe how audio data is stored, with many formats compressing the audio data resulting in smaller files size. Audio can be compressed, resulting in a small file; this is however at the expense of audio quality (lossy compression). There is a huge choice of audio formats - MP3, WMA, Wave and Ogg Vorbis. The files used for the experiment were a collection of MP3 audio files downloaded from the internet (mostly samples from music publishers) as well as sound bytes from commercial internet sites.

4.2.8 Program source code (a sample of the Open Source applications source code files)

Source code files were mostly C++, some Perl and C application programs. Source code files are ASCII text files, the difference between source code files and other text files is that source files are machine readable; a parser parses the source and converts
it from English-like form to a form the computer can read and execute. Computers require binary strings of 1’s and 0’s that mean very little to human readers, but can be very quickly and accurately understood by the computer. The English-like program is called the “source code”, and the resulting compiled code produced by the compiler is usually called an “object file”. One or more object files are combined with predefined libraries by a linker, to produce a final complete file that can be executed by a computer. A library is a collection of pre-compiled “object code” that provides operations that are performed repeatedly by many computer programs. As source code files are machine readable they are very fragile, the smallest error often rendering source code files unusable. Source code files can only be compiled when completely error free.

4.3 Introducing noise to the archived documents

The experiment introduced noise to the archival collection artificially by means of a simple program written for this purpose. The program simply opened each file in the collection in read mode, and then proceeded to read the contents then write the contents to a modified file while introducing noise at a predetermined level. The process occurred in a loop and terminated when the end of the file was reached, so for a 3% noise introduction every thirty third character was substituted by multiplying it by “-1” effectively flipping its bits; for a 7% noise introduced every 14th character was subjected to similar treatment, and so on. The process described above means the noise introduced was evenly distributed within the digital documents and throughout the archive.
4.4 Noise distribution

While naturally occurring noise from the passage of time is not evenly distributed as is the artificially introduced noise in this research, examining a large number of files with errors failed to reveal any pattern that can be replicated for the purpose of this experiment. It is observable on removable media that the damage is often contiguous for example scratches on optical media, tears to tape, and head crashes in system disks, which are all caused by mechanical damage. This experiment however is concerned with all kinds of damage not limited to mechanical causes. Aside from mechanical damage, media used for archival storage is susceptible to damage when exposed to strong magnetic fields, exposure to direct sunlight, storage in unfavourable conditions (humidity, high temperature), and material decay (for instance optical media suffer from oxidisation issues). The nature of introduced errors varies according to the nature of the damage, as the nature of noise introduced varies from contiguous noise resulting from mechanical / physical damage, to noise resulting from adverse conditions such as poor storage conditions.

The artificially introduced noise types possible also varied from additive noise to Gaussian noise to burst noise. This is not unlike the noise effects used in the field of image processing; Image manipulation software packages contain operators to artificially add noise to an image. In the absence of a recognized model of noise distribution, for data stored under different conditions, for different length of time, the research artificially introduced noise at regular interval and of regular size, this is a similar noise distribution scheme to one used in field of image manipulation. The scheme is referred to as salt-and-pepper (or speckle) noise distribution as used in image processing software, this noise model is described in Section 4.1.3.1 below.
4.4.1 Speckle (Salt and Pepper) Noise

Speckle noise is a common form of noise (data drop-out noise commonly referred to as intensity spikes, speckle or salt and pepper noise). This noise is caused by errors in the data stream. The corrupted bits are either set to the maximum value or have single bits flipped. In some cases, single bits are set alternatively to zero or to the maximum value, giving the stream ‘salt-and-pepper’ like spikes. The noise is usually quantified by the percentage of corrupted bits.

4.4.2 Varied amounts of artificial noise

The absence of a distribution model discussed in Section 4.1.3 lead to the decision to adapt a noise distribution model from the field of image processing and to use four different levels of artificially introduced noise in four experimental iterations. The lower levels were intended to simulate various amounts of distributed damage resulting from adverse storage conditions. Higher amounts of noise were intended to mimic more severe damage; higher amounts of artificially introduced noise while not contiguous, due to the more frequent recurrence of the distortion (1 in 14) mimicked damage resulting from mechanical / physical damage to the media.

4.4.2.1 Adapting an image processing noise model

Real world stored bit-streams usually contain departures from the bit-stream that are produced by output devices during the storage process. Such departures are referred to as noise. Noise arises as a result of un-modelled or un-model-able processes
occurring post the storage process. The noise is not part of the archival documents written to a storage medium and is caused by a wide range of sources, such as variations in the magnetic media's magnetic field, environmental conditions, physical damage by mechanical means, device errors, and so on. The characteristics of these departures (noise) depend on the cause and its severity.

Deliberately corrupting an archive with noise allowed the research to test the resistance of different file types to noise and assess the performance of various Open Source software tools used for this experiment.

4.5 Initial observations

It was found that the higher the percentage of introduced noise the greater the adverse impact on the stored archive. Tests were conducted with 3%, 7%, 11% and 18% of introduced noise. The research experiment was conducted introducing artificial noise at these levels in order to mimic the effects of time on the digital archive. These levels were chosen so as to attempt to simulate a range of damage types from damage occurring due to mechanical/physical media damage, to damage occurring due to adverse storage conditions. Noise levels at 3% were found to have little or no observable effects on all but source code files. It is worthy of note that a single bit change in an executable file renders it useless, however the experiment as outlined in Chapter 3 stores no binary executables, only source code (to be compiled by users to executable applications). It is for this reason - zero tolerance to noise - that binary executables were not considered for storage; by contrast storing source files provides the robustness in this archival approach. Results for the tests conducted are summarised in Tables 4.2, 4.3, 4.4, and 4.5. No experiments were performed with
higher than 18% noise; the reason being that at 18% the damage to stored files was so extensive as to render all but a few files usable.

Table 4.2 Percentage of damage observable at a 3% level of artificially introduced noise.

<table>
<thead>
<tr>
<th>File type</th>
<th>3.00% Lossiness</th>
<th>3.00% Editability</th>
<th>3.00% Saving</th>
<th>3.00% Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doc+Embed Cyrillic</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Doc+Emb. Graphics</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Doc+Emb. Images</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Doc+Emb. Hyperlinks</td>
<td>7.50</td>
<td>0.00</td>
<td>0.00</td>
<td>5.00</td>
</tr>
<tr>
<td>Doc+Emb. Tables</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Text only</td>
<td>5.00</td>
<td>0.00</td>
<td>0.00</td>
<td>5.00</td>
</tr>
<tr>
<td>Spreadsheets</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>PDF files</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>MPEG4 video</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>.mov &quot;Trailer&quot;</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Jpeg images</td>
<td>7.69</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Audio files</td>
<td>10.00</td>
<td>0.00</td>
<td>0.00</td>
<td>100.00</td>
</tr>
<tr>
<td>Source code</td>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
<td>100.00</td>
</tr>
</tbody>
</table>

Table 4.3 Percentage of damage observable at a 7% level of artificially introduced noise.

<table>
<thead>
<tr>
<th>File type</th>
<th>7.00% Lossiness</th>
<th>7.00% Editability</th>
<th>7.00% Saving</th>
<th>7.00% Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doc+Embed Cyrillic</td>
<td>1.25</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Doc+Emb. Graphics</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Doc+Emb. Images</td>
<td>1.18</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Doc+Emb. Hyperlinks</td>
<td>2.50</td>
<td>0.00</td>
<td>0.00</td>
<td>2.50</td>
</tr>
<tr>
<td>Doc+Emb. Tables</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Text only</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Spreadsheets</td>
<td>5.00</td>
<td>0.00</td>
<td>0.00</td>
<td>10.00</td>
</tr>
<tr>
<td>PDF files</td>
<td>5.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>MPEG4 video</td>
<td>5.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>.mov &quot;Trailer&quot;</td>
<td>60.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Jpeg images</td>
<td>23.08</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Audio files</td>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
<td>100.00</td>
</tr>
<tr>
<td>Source code</td>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
<td>100.00</td>
</tr>
<tr>
<td>Lossiness</td>
<td>Editability</td>
<td>Saving</td>
<td>Functionality</td>
<td>File Type</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>--------</td>
<td>--------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>11.00%</td>
<td>11.00%</td>
<td>11.00%</td>
<td>11.00%</td>
<td></td>
</tr>
<tr>
<td>17.92</td>
<td>4.17</td>
<td>17.08</td>
<td>0.00</td>
<td>Doc+Embed Cyrillic</td>
</tr>
<tr>
<td>23.81</td>
<td>0.00</td>
<td>57.14</td>
<td>0.00</td>
<td>Doc+Emb. Graphics</td>
</tr>
<tr>
<td>2.35</td>
<td>2.35</td>
<td>0.00</td>
<td>0.00</td>
<td>Doc+Emb. Images</td>
</tr>
<tr>
<td>10.00</td>
<td>0.00</td>
<td>0.00</td>
<td>10.00</td>
<td>Doc+Emb. Hyperlinks</td>
</tr>
<tr>
<td>3.13</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>Doc+Emb Tables</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>Text only</td>
</tr>
<tr>
<td>25.00</td>
<td>0.00</td>
<td>0.00</td>
<td>25.00</td>
<td>Spreadsheets</td>
</tr>
<tr>
<td>35.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>PDF files</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>MPEG4 video</td>
</tr>
<tr>
<td>50.00</td>
<td>50.00</td>
<td>50.00</td>
<td>0.00</td>
<td>.mov &quot;Trailer&quot;</td>
</tr>
<tr>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>Jpeg images</td>
</tr>
<tr>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
<td>100.00</td>
<td>Audio files</td>
</tr>
<tr>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
<td>100.00</td>
<td>Source code</td>
</tr>
</tbody>
</table>

Table 4.4 Percentage of damage observable at an 11% level of artificially introduced noise

<table>
<thead>
<tr>
<th>Lossiness</th>
<th>Editability</th>
<th>Saving</th>
<th>Functionality</th>
<th>File Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.00%</td>
<td>18.00%</td>
<td>18.00%</td>
<td>18.00%</td>
<td></td>
</tr>
<tr>
<td>20.83</td>
<td>20.83</td>
<td>20.83</td>
<td>0.00</td>
<td>Doc+Embed Cyrillic</td>
</tr>
<tr>
<td>100.00</td>
<td>61.90</td>
<td>100.00</td>
<td>0.00</td>
<td>Doc+Emb. Graphics</td>
</tr>
<tr>
<td>31.76</td>
<td>34.12</td>
<td>3.53</td>
<td>0.00</td>
<td>Doc+Emb. Images</td>
</tr>
<tr>
<td>27.50</td>
<td>7.50</td>
<td>0.00</td>
<td>27.50</td>
<td>Doc+Emb. Hyperlinks</td>
</tr>
<tr>
<td>18.75</td>
<td>3.13</td>
<td>6.25</td>
<td>0.00</td>
<td>Doc+Emb Tables</td>
</tr>
<tr>
<td>9.09</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>Text only</td>
</tr>
<tr>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
<td>25.00</td>
<td>Spreadsheets</td>
</tr>
<tr>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>PDF files</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>MPEG4 video</td>
</tr>
<tr>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
<td>.mov &quot;Trailer&quot;</td>
</tr>
<tr>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>Jpeg images</td>
</tr>
<tr>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>Audio files</td>
</tr>
<tr>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
<td>100.00</td>
<td>Source code</td>
</tr>
</tbody>
</table>

Table 4.5 Percentage of damage observable at an 18% level of artificially introduced noise

4.5.1 The Data Collected

Tables 4.2 to 4.5 present the data collected at each iteration of the experimental process. Analysing these results is the subject matter of this chapter. The columns represent the percentage of damage observed after introducing noise. The header row indicates the level of noise introduced and the criteria measured in the column below,
Chapter 3 outlined the criteria that the experiment assessed and the reason for choosing each particular criterion. The criteria are lossiness, edit-ability, saving (ability to), and functionality retention. The disparity in the sample size for each file type was the reason percentages were used as opposed to raw numbers (and results were rounded to 2 decimal places). The file types used in the experiment are hereunder described.

4.6 Results

The results of the experimental process outlined in Section 4.2 and in Tables 4.1 through 4.4 were converted to bar charts which depict the frequency of adverse noise impact as the independent variable (y-axis) and the type and percentage of files affected as the dependent variable.

![Figure 4.2: Effects of 3% artificial noise](image)

4.6.1 Effects of 3% Introduced Noise

Figure 4.2 charts the effect of 3% artificially introduce noise on the archive. The results indicate that all but source code files displayed resilience / tolerance to low
levels of distortion. All files opened successfully in Open Source software tools and discernible damage was noted; 1 audio file out of 13 failed to play back. Source code files on the other hand were all adversely affected by the introduced noise - all failed to compile.

![At 7% noise](chart.png)

**Figure 4.3: Effects of 7% artificial noise**

### 4.6.2 Effects of 7% Introduced Noise

Figure 4.3 charts the effect of 7% artificially introduce noise on the archive. The results indicate that 5 .doc files out 456 had discernible errors, 1 in 20 spreadsheets as well as 1 in 20 PDF files had errors, 12 in 20 jpg files had visible errors, 3 audio files failed and all source code files had compile errors. All files opened successfully in Open Source software tools and discernible damage was noted.
4.6.3 Effects of 11% Introduced Noise

Figure 4.4 charts the effect of 11% artificially introduce noise on the archive. The results indicate that 56 .doc files out 456 had discernible errors, 5 in 20 spreadsheets as well as 7 in 20 PDF files had errors, 15 in 20 jpg files had visible errors and 5 failed to open with an error message “This is not a jpeg- jpg file”, 13 audio files failed. 2 of 4 .mov movie trailer failed to run, and all source code files had compile errors.

Figure 4.5: Effects of 18% artificial noise
4.6.4 Effects of 18% Introduced Noise

Figure 4.5 charts the effects of 18% artificially introduce noise on the archive. The results indicate that 105 .doc files out of 456 had discernible errors and a further 20 failed to open in Open Office, 20 in 20 spreadsheets as well as 20 in 20 PDF files had errors, 17 spreadsheet files failed to open as did 8 PDF files, 7 in 20 .jpg files had visible errors and 13 failed to open with an error message "This is not a jpeg-.jpg file", 4 .mov movie trailers failed, 13 audio files failed and all source code files had compile errors.

4.7 Aims of the experiment

The experiment carried out the aim stated in Chapter 1 "To explore a robust, low cost, low overhead, and widely available method for long term digital document preservation that is capable of transcending system and software application obsolescence, while remaining capable of delivering to future users the digital documents stored as their original authors saw them". Furthermore it also demonstrated that storage capacity in modern networks can be effectively used for archival purposes. Unused storage capacity within networks is growing exponentially with each network upgrade. In the case of the experimental cluster the unused storage capacity was in excess of 1 terabyte, as outlined in Section 3.2.

The experiment demonstrated that the simpler the file content format is of a given archival file, or the more flexible the decoding and viewing mechanism is, the greater the chance that this file type is likely to survive the test of time with little or no discernible adverse effects.
4.8 Observations and arguments

The observable effect on other files indicates that a correlation exists between the functionality and complexity of the file format and its resilience to introduced noise, as demonstrated by the consistent adverse effect irrespective of the amount of noise contained within the source code files. Without error correction / recovery mechanisms, source code files are not useable without a non trivial debug effort on the part of future users. The research examined the long term archival survival success measures developed in Chapter 3, namely edit-ability, lossiness, saving, and functionality. The data collected demonstrated the fragility of digital files and their susceptibility to damage. The research also demonstrated the viability of Open Source tools as means of accessing files created by proprietary software.

4.9 What of the compressed archives

No experimental work was done with the compressed archive. The compressed archive as stated in Section 3.11 was used as an error recovery mechanism, the RAR compression and decompression algorithm have documented tolerance to errors up to 10%. When compressed with appropriate options the RAR de-compressor can rebuild 1 lost volume in 10 of a fragmented archive. The RAR archive format supports redundant recovery record. (RARLAB, 2004)

When an archive is created containing a recovery record, it can be repaired in the case of physical data damage due to disk failure or data losses of any other kind. The recovery record contains up to 524288 recovery sectors. If data are damaged continuously, then each recovery sector is able to recover 512 bytes of damaged information. This value may be lower in cases of multiple damage.
The recovery record size may be specified at the time of creating the archive; it is usually expressed as a percentage of the archive size. While the recovery record increases the size of an archive it also increases its resilience and recoverability. The size of the recovery record may be approximated using the formula:

\[(\text{Archive size}/256 + \text{number of recovery sectors}) \times 512 \text{ bytes}\]

Recovery volumes allow the reconstruction of missing or damaged files in a volume set (a multi volume archive). This feature was especially suited for this research. Each recovery volume is able to reconstruct one missing RAR volume. For example, if an archive comprises 30 volumes and 3 recovery volumes, it is possible to reconstruct any 3 missing volumes. If however the number of .rev files is less then the number of missing volumes, reconstruction is impossible. The total number of usual and recovery volumes must not exceed 255 and the number of recovery volumes must be less than the number of RAR volumes. (RARLAB, 2004)

Error recovery is an integral part of any archiving effort, there are many Open Source compressors available, and while the RAR compressor is not Open Source the decompressor is available as Open Source, and as such complies with the experimental premise. The advantage of using RAR compression over Open Source compressors is its documented error recovery discussed above. In an effort to provide maximum longevity to the archive, the decision to use RAR for its redundant error recovery mechanism is well justified. Compression in the context of this research was not for the purpose of space saving; indeed the compression options used reduced the overall space saving outcomes possible by using a compressor. The trade-off was higher error tolerance.
Chapter 5 DISCUSSION

5.1 Findings

The experiment result data revealed a correlation between the amounts of introduced noise and the number of adversely affected files, except in the case of source code, where all files were adversely affected. Furthermore, the charts (Figures 4.2 to 4.5 inclusive) revealed a correlation between the amount of introduced noise and the severity of its adverse effects (again the exception is in the case of source code files).

Closer examination of these results revealed that the complexity of the file format and the specific file use significantly reduced the tolerance to introduced noise (for example spreadsheet files with their multiple worksheets and embedded formulas, similarly machine readable files such as source code or binary executable files), while other file types such as simple text files and MPEG files remained viable at the maximum experimental noise levels. Sections 5.1.1 through 5.1.4 discuss the effect of introduced noise in more detail.

5.1.1 The 3% results

As mentioned in section 4.6.1 at a level of 3% introduced noise, 5% of the spreadsheet files lost functionality, and 100% of source code files were rendered faulty.

- Source code files are machine readable, as outlined in Section 4.4 code compilers are intolerant of errors as they translate the source code into binary
machine instructions.

- Spreadsheets on the other hand appear to be more tolerant to faults; the spreadsheet file format is in coma or tab delimited values. A change or introduced noise effect is limited to the value altered occasionally the change coincides with the location of a formula (a predefined relationship to the other values as described in Section 4.2.2) and that formula fails. More severe errors are encountered when the noise affects the delimiter and disturbs the general structure of the file.

- MPEG file are more tolerant than most file to noise, the structure and nature of movie files renders more resilient to introduced noise, when the human eye perceives a movie, it perceives a sequence of images in which objects appear at a sequence of positions. Although each frame represents a frozen instant of time, the movie gives a convincing impression of motion. Somehow the visual system interprets the succession of still images so as to arrive at a perception of a continuously moving scene (Adelson, 1985, p-1). The file structure thus consist of a series of picture frames to be rendered at a rate determined by the encoding algorithm, along with a sound, introduced noise alters a number of frames in the sequence, but when the sequence is rendered the very phenomenon described previously by Adelson makes these alteration imperceptible to the human eye, short of a frame by frame examination of the file introduced noise does no discernible damage to the movie in question.
5.1.2 The 7% results

At a level of 7% of introduced noise, 2.5% of the .doc files containing hyperlinks lost functionality (in 2 of 40 files, the hyperlinks failed).

- Hyperlink in .doc files are pointers to a location either on the local host or a remote host, when introduced noise alter the address pointed to by the pointer it renders it invalid and thus this functionality becomes lost.

100% of source code files were rendered faulty by 7% of introduced noise.

Spreadsheets on the other hand appear to be more tolerant to faults: two files failed. As discussed earlier the effect of introduced noise is limited to the value altered and occasionally the change coincides with the location of a formula (a predefined relationship to the other values as described in Section 4.2.2) and that formula fails.

60% of Jpeg files were affected in terms of lossiness the quality of the image displayed was degraded, while the Jpeg algorithm uses approximation to render the attribute of the image the data loss causes the approximation to be less accurate than would normally otherwise be the perceptible outcome is a poorer quality rendering of the image. Also 2 Jpeg files failed to open. It is worth noting that the Jpeg files that failed to open using the Gimp image processing application selected in Section 3.9.3, displayed when opened using a web browser, the browser used being “Galleon” - a “Gecko” based Open Source browser (not too dissimilar to Netscape 7.0). The quality of these files when viewed in the “Galleon” browser was degraded, the colours and sharpness of detail becoming severely degraded.
5.1.3 The 11% results

At a level of 11% of introduced noise, 56% of the .doc files were compromised and failed to open.

- Three such files, opened after several error warnings were dismissed but displayed no contents apart from some non printable characters.

- The Cyrillic text in a further 17% of the files was substituted by “?” while the English text persisted.
  - At this level of distortion secondary encoding information is compromised and the Cyrillic characters are lost, the saving ability of these files was also affected as saved files when reopened contained errors and “?”.

Once again 100% of source code files were rendered faulty by 11% of introduced noise, as to be expected.

Spreadsheets on the other hand were more tolerant to faults where, at this level, 25% of the files failed. As discussed earlier the effect of introduced noise is limited to the value altered and occasionally the change coincides with the location of a formula (a predefined relationship to the other values as described in Section 4.2.2) and that formula fails.

100% of Jpeg files were affected in terms of lossiness the quality of the displayed image was degraded, and 8 files failed to open. It is worth noting that the two Jpeg files that displayed when opened using a web browser at 7%, failed to open in the web browser at 11% introduced noise.
50% of the `.mov` files were adversely affected; edits to the files to remove a section of the trailer were unsuccessful, and attempts to save the files also failed, the files were able to be played back. 100% of the audio files were also adversely affected by 11% of introduced noise.

### 5.1.4 The 18% results

At a level of 18% introduced noise the damage was widespread amongst the archival collection. The `.doc` files were compromised as follows; 20% of the `.doc` containing Cyrillic text, the Cyrillic text was substituted by `"?"`, while the English text persisted. At this level of distortion secondary encoding information is compromised and the Cyrillic characters were lost. The ability to save these files was also affected. Of the 17% mentioned, 4.17% of these files when opened displayed as “read only”. The most resilient files - namely text only documents - had a 9.09% error rate, formatting within the document being affected. The source code files had a 100% failure rate, these were rendered faulty by 3% of introduced noise and at 18% the failures were only worse the number of compile errors exceeded the compiler tolerance and it quit.

Spreadsheets had up to this point displayed some resilience to faults (as discussed in Section 5.1.3 some 75% of spreadsheets remained viable at 11% introduced noise) at 18% introduced noise all 100% of the file were affected and 25% of the spreadsheet files failed to display.

100% of Jpeg files were affected - all files failed to open.

100% of the `.mov` files were not able to be played back.

100% of the audio files failed.
The MPEG video file used in the experiment played back at all levels of introduced noise with no discernible adverse effect. It is worth noting that at 25 frames per second it is unlikely that the human eye will discern any lossiness in the quality of some frames or even the total absence of some frames. Persistence of vision starts at 12 frames per second; simple mathematics infer that at levels exceeding 40% noise, noticeable lossiness may be discernible.

5.2 Summary of Results

Closer examination of the results revealed that the complexity of the file format and the specific file use significantly reduced the tolerance to introduced noise, while other file types remained viable at the maximum experimental noise levels. The files that withstood the most noise were characterised by one of 2 characteristics:

(i) Simple encoding (simple text, or plain text documents)

(ii) High levels of compression and approximation in the decompression algorithms

The source code files, although compromised as machine readable files, were still human readable. The ASCII text of the code was still readable at 18% of introduced noise, likewise the MPEG still played back. The reason is that in both cases the viewer was error tolerant; the human eye reads the text of the code with error, and with some learning on behalf of the reader he/she can correct the errors and render the code compile-able again. The MPEG decoder uses approximation in decompressing the image data and as such is tolerant of errors in the files. Short of the total loss of file headers which contain information as encoding and frame rate,

S. M. Antoun

90

Chapter 5
the decoder will approximate the image displayed from the files it reads. The speed (number of frames per second) means that if one or more frames are compromised, the human eye will not discern the compromise. If however the images were to be analysed one frame at a time the lossiness will be detected.

5.3 Could proprietary software perform achieve similar outcomes?

Section 3.6 discussed the question: Could proprietary software fulfil the task Open Source software does in this research? The simple answer was “no” the reasons being:

- Proprietary software source code is seldom published,
- Lehman’s second law observes that the structure of evolving software will degrade over time unless remedial action is taken; in the absence of access to the software source this remedial action is never undertaken (Bennet, 1995, p20). This often is the case where the software developers go out of business.
- License / patent owners of the software provide a license to use only;
- The license obtained by purchasing the software precludes copying the software
- The license permits only one installation.
- Archiving a collection for extended periods does not mean this license condition will not be breached in the future, and
- The license does not permit users to modify the software which is sold as
binary executables only, and software licences prohibit reverse engineering (executable files cannot be modified, in order to modify an executable program the machine readable source code must be modified).

The task of recovering stored physical records from fragments involves the use of novel methods and adapting existing technology to the task of recovery, the adapting of software tools available only in binary executable formats would involve reverse engineering the tool (1\textsuperscript{st} violation of End User Licence Agreement (EULA)), modifying the tool (2\textsuperscript{nd} violation of EULA), and recompiling the modified application to perform the modified functionality (3\textsuperscript{rd} violation of EULA). Reverse engineering a large application is no simple task, outcome is the assembler source code which by nature is not a simple programming language an modifying existing application assembler source is a complex if not all together an impossible task.

5.4 Backup and Compressed Volumes

The experiment also used a back up compressed archive, to test the viability of compression algorithms in error recovery. Of all the different tools available Rar compression was found to be most suited to this experiment for 2 reasons, first the decompressor is available as an open source application under the GPL licence, secondly Rar has superior error recovery ability documented error tolerance to a maximum 10\% loss / damage. Rar is able to reconstitute 1 in ten volumes of a multi-volume archive. For these reasons Rar was used as the experiment error recovery mechanism and when tested it performed as documented in its documentation.
5.5 Authenticity and Accuracy

Issues of authenticity and accuracy were not addressed by this research, the research focused on the delivery of the archive, the applications, and the technology behind the applications (source code) to future users in order to provide them with the means to view the documents as their authors saw them at the time of creation. Issues of authenticity and accuracy of the archival documents were beyond the scope of this research as outlined in chapter 3 research methodology.

5.6 New Digital Environments

The research considered records produced in new digital environments, experiential, dynamic, and interactive, and objects resulting from artistic, scientific and government activities. The very premise of using a collection of Open Source software is to permit users maximum flexibility to use any tools they want to render a creative achievable output, compared with the other archiving models discussed in Chapter 2.

5.7 Research Findings

The research presented the contingency of last resort; the project provided sufficient data to assist in providing viewing tools for future users to display the stored digital documents, if and when other archiving methods fail. This approach was designed to be non restrictive, allowing the use of any authoring tool available at the time to create digital documents. The research described a viable method for archival storage,
and a viable approach to dealing with issues of viewing and authoring software obsolescence.

Demonstrating the validity of this method, this thesis presented an experiment where Open Source application source code was stored along with the digital archive. The only requirement was a suitable compiler for the system to compile the application(s). This resulted in an application to view the document that was compatible with the users' platform and the archived data.

The research demonstrated that Open Source applications are indeed capable of transcending the divide between operating platforms. The research argued the viability of Open Source applications in preserving data for medium- to long-term, based on the adaptability of the software. The source code is stored along with the data; when the operating environments evolve to become unsuited to the application, the first option is for the source to be ported to the new operating system environment. The alternate option is to develop a system-compatible compiler to compile the stored source code. It is worth noting that both options require a non trivial effort on the part of future users; however option two is a once only effort as the compiler can be used repeatedly to compile multiple applications.

In simulating the effect of the passage of time, the experiment introduced distributed noise to the archival collection. The amounts of introduced noise was increased with every iteration of the experiment in order to determine the degree of resilience / fault tolerance of different file types in the archival collection, and to mimic the various outside factors that may affect the archival collection. These range from physical mechanical damage of the storage media, to introduced departures from the original
bit stream caused by: device faults, media deterioration, and poor storage conditions. The lower noise levels were intended to simulate various amounts of distributed damage resulting from adverse storage conditions, while higher amounts of noise were intended to mimic more severe damage. Higher amounts of artificially introduced noise while not contiguous, (due to the more frequent recurrence of the introduced noise 1 in 14) mimicked damage resulting from mechanical / physical damage to the media (i.e. scratches to disk surface, head crashes, outside magnetic field damage to magnetic media).

During the cyclical iteration of the experiment with introduced noise, the need for error recovery in archival storage was noted. The research used a proprietary compression tool to compress a redundant copy of the archival collection. While at first glance this seems to contradict the research premise of using Open Source software, in fact it does not as the decompression tool is an Open Source tool. The decompression tool is the only tool required to retrieve and view the archive contents. This being the case, the use of compression is within the project parameters. It is also worthy of note that the compression tool was chosen for the superior error recovery, as described in Section 4.9.

The research has demonstrated the viability and flexibility of the Open Source approach. Users are afforded maximum flexibility as to file type and application of choice as opposed to VERS which restricts the users to a limited pool of case specific tools to use as discussed in Section 4.7.
Chapter 6 CONCLUSIONS

Today's digital document is not immune from loss, but whereas early material was capable of being partially recovered from physical fragments (sections of stone monument, clay tablet, partial print copy etc...), a bit stream of zeros and ones cannot be intelligibly recovered from fragments!

6.1 Open Source and Archival Survivability

The research premise that Open Source tools' source code can be modified by users to adapt it to their platform is unquestionable. The very existence of Open Source is testimony to this assertion. The Open Source tools in existence today exist because code contributors downloaded the source of older versions modified them and ported them to newer platforms. Code contributors ported Open Office from "Solaris™" operating environment to the Windows™ operating environment (all version above 3.1). The portability of the source code from one platform to the next is indeed a non trivial effort but is required only once to render countless archives accessible and viewable as seen by their authors. This research in its entirety (except for screen dumps from Window™ environment and tools) was conducted using Open Source tools and operating system. The initial learning curve for novice users of Open Source environments and tools is steep. Open Source tools are written by "expert" / "advanced" programmers who from time to time, assume a certain amount of savvy on the part of the users. This is most discernible in help files, which are often written in the language of online manual pages - in other words, terse, with little if any examples, and with little regard for context, simply information (and it is left up to the users to interpret it).
The research demonstrated the viability of Open Source tools and operating systems. The users are afforded maximum flexibility as to file type and application of choice (compared with VERS, which restricts users to a limited pool of case-specific tools).

- This research is premised on users storing the archive files along with suitable Open Source tools to access them.

- It is true that not every proprietary application has an Open Source equivalent but the majority of popular proprietary applications do. It may be necessary for some users to become active participants in the Open Source project and initiate an effort to build an application equivalent to the proprietary application they are using in order to provide their intellectual / creative output with means of long term survival.

- The research has demonstrated the viability of its stated aim “To explore a robust, low cost, low overhead, and widely available method for long term digital document preservation that is capable of transcending system and software application obsolescence while remaining capable of delivering to future users the digital documents stored as their original authors saw them”.

  - Unlike similar efforts such as “Simulating an operating system on another” (Rothenberg, 2000), “Preserving functionality in a digital archive” (Oltman, 2004), “Buckets digital technology” (Nelson, 2001) and “VERS” (Waugh, 2001), is not dependant on hardware, or purpose built software tools. The Open Source approach is one that is characterised by its portability from one platform to the next, this ability provide its with a unique survivability mechanism that allows it to transcend changes in Operating environment as well as changes to...
the hardware current at the time of retrieval.

- While the effort to port the software from one platform to the next is non-trivial it is a once only effort. The software tools once adapted to the new environment can be used to render the entire archival collection it relates to. By comparison to the effort required to adapt each file in an archive to new software is a greater burden as is the case in Oltman’s approach, or the near total loss that would result in the case of metadata loss approaches such as VERS or Buckets technology.

- This does not supplant the need for using a number of redundant copies on removable media. The experimental result indicated a simple fact: the more distortion a file is exposed to the greater the chance it will fail. Redundant copies of an archive can serve as error recovery mechanism to provide maximum longevity to the digital collection.

6.2 Future directions

It would be naive to think that the archiving problem has a single solution (i.e. panacea). The logical next step towards a reliable long-term archival mechanism must address media issues as well as technology obsolescence issues. While competition between manufacturers continues to drive the technology evolution, these issues will remain unresolved.

The results of this study suggest that using Open Source applications to reproduce the
documents authored by proprietary tools within a given operating system on different platforms offers a way of displaying the document as the original software did at the time of authoring. Given what has been outlined with regards to the portability of Open Source, users in the distant future can thereby recreate the content, functionality, and usage (editing) of the original document.

The results of the experiment described indicate that this approach should work in principle, assuming that suitable compilers for source code can be developed for future platforms. It is recommended that additional iterations of this experiment be performed in a longitudinal study, as described above. This would allow more realistic demonstrations of the effectiveness of the Open Source approach as a preservation method.

If a longitudinal series of such experiments were to be performed on archived material over a number of years as suggested, and if no major surprises are encountered in performing them, it is likely that a viable, long-term solution to the problem of digital preservation can be developed using Open Source tools which could enable document-compatible application software to be run indefinitely.
Glossary

The major concepts used in this study are briefly outlined in this section. Other terminology and preservation concepts are described in appropriate places throughout the thesis.

**Digital document**: Formally: A computer file when opened with the appropriate software displays the intellectual output as authored by its creator. This may be a graphic image, a sequence of musical notes, a video animation, a text document or any other sequence of digits arranged by its author to be intelligible when appropriately displayed.

**Archive** is a repository of documents and files conserved for use at some future point in time. A long term archive is such a repository where the future use is envisaged to occur in a number of years, decades, or even centuries into the future.

**Media** are the physical objects upon which digital documents are written / saved. Storage media can mean a variety of materials and devices, for example magnetic disks, magnetic tapes, optical disks, paper tape, punched cards, and vinyl disks.

**Storage devices** are the devices used to write to and read from media. These can include magnetic disk drives, optical drives, tape drives, and analogue recorders / players.

**Internet** is a large public computer network that connects people around the world. It is a common medium for personal, academic, and business communications. The Internet has become an easily accessible network connecting businesses to customers, researchers to information, and people to people.

**Long-term preservation** means two distinct but equally important functions: long-
term maintenance of a byte stream and continuing access to its contents through time and changing technology (Beagrie, 2001, p5).
Bibliography:


Bibliography
Kahle, Brewster, 1997, “Archiving the Internet” Scientific American March
(http://www.archive.org/sciam_article).

Guide for CD-ROM Drives and Compact Disk Media"

Research Centre, JCDL'01, June 2001.

and Development Activity” Department of Information Science Loughborough
University JCDL.

National Archives of Australia "RecordKeeping a New Approach"
9/12/02).

Nelson, Michael L. 2001, "Buckets a digital technology for preserving NASA

Oltmans, Erik, van Diessen, Raymond J., van Wijngaarden, Hilde. 2004,
"Preservation Functionality in a Digital Archive" Proceedings of the 2004 Joint

OpenOffice.org “build instructions”
http://www.openoffice.org/dev_docs/source/build_linux.html#BuildingaFullBuildof
OpenOffice Last visit October 2002.

Open Source Initiative (OSI), The Open Software License 2.0, and The Academic
Free License 2.0, March 2004, http://opensource.org/licenses/ (last visited March
19, 2004).

Open Channel Software, NASA COSMIC Collection 2004 at the Open Channel

S. M. Antoun 104 Bibliography

http://www.acnc.com/04_01_01.html last visited 18/07/03.


RARLAB knowledge base, Recovery Volumes 

Ross, Seamus, Gow, Ann, 1999 “Digital Archaeology: Rescuing Neglected and Damaged Data Resources A Study within the Electronic Libraries (eLib) Programme on the Preservation of Electronic Materials” Library Information Technology Centre, South Bank University, pp 1-83.


SourceForge 2004 "Open Source Development Network website", 
http://sourceforge.net last visited January 2004, and 


S. M. Antoun 105 Bibliography


Appendix I  Building an application from source

In this section is a non exhaustive step by step guide to building applications from source, the application is OpenOffice.org an Open Source MS Office equivilant, the source code should be is a directory created for the purpose and throughout this section will be referred to as: "$SRC_ROOT".

The configuration script should be run; it will check all requirements and create the script LinuxIntelEnv.Set.

Running the LinuxIntelEnv.Set script sets all environment variables.

Build tools should be created using bootstrap.

The application is built by using the command dmake in $SRC_ROOT,
or build -all in the "instsetoo" module,
or build followed by deliver in the individual modules.

Building an Individual Project (Tool)

1. (These are instructions supplied with the source, the test compile carried out was for the entire package no individual tool build attempts have been made)

2. To build an individual project, the steps are:

3. Check out the modules for the project that to build from the OpenOffice.org cvs tree. If project uses the resources module, res, check out this module also.

4. Check out the module config_office. This is always necessary to create the build environment.

5. Also check out the modules dmake, external, xml2cmp, mkdepend, sal, solenv, vos, tools and rsepp. These are required by bootstrap.

6. Unpack the solver tarball in the $SRC_ROOT directory.
7. Run the configure script to check all requirements and to create the script LinuxLitelEnv.Set.

8. Run the script LinuxIntelEnv.Set to set all environment variables.

9. Create the build tools using bootstrap.

10. Build each module against the prebuild solver using the build tool, followed by deliver.

**Build Requirements**

Before building, ensure that the system satisfies the recommended software and minimum hardware requirements for the type of system being worked on.

**Software Requirements**

* glibc 2.1.x or higher

* gcc: OpenOffice.org has been successfully built under Linux using the gcc versions 3.0.x, 3.1.1, and 3.2.1. Older versions were built with gcc 2.95.2. Version 2.96 does not work!

* The X11 development libraries and header files should be installed.

* PAM (should come with most Linux distributions) JDK 1.3.1

* Perl 5

* csh

* zip and unzip

* The gpc general polygon clipper library release 2.31, the files gpc.c and gpc.h in $SRC_ROOT/external/gpc.
Hardware Requirements

* Intel Pentium II
* 128 MB RAM recommended
* 3 GB free disk space

The source code

oo_643B_src.tar.gz (In this case is the 643B release being used for this research)

Unpack the tarballs as follows:

> gunzip oo_643B_src.tar.gz
> tar -xvf oo_643B_src.tar
> cd oo_643B_src

Generating the Build Environment and Build Tools

Use the configure script to generate the build environment. The configure script checks that all software, hardware, and system requirements for the build are satisfied, and it creates a configuration file called LinuxIntelEnv.Set that the source command is then run on to set all necessary build environment variables.

This configuration file will be moved into the SRC_ROOT directory. A top-level makefile script makefile.mk and the script bootstrap in the config-office directory will be moved into SRC_ROOT as well. This is due to technical reasons: The SRC_ROOT directory in the cvs tree can only hold directories. On the other hand, the top-level makefile.mk should logically be placed in the top-level directory SRC_ROOT. The cvs tree holds these files in config Office and configure copies them up.
Before running configure, the environment variables CC and CXX must point to c and c++ compiler:

```
CC=/path/to/gcc
CXX=/path/to/g++
```

To run the configure script, the following command is used:

```
$SRC_ROOT> cd config_office
config_office> ./configure
```

There are a number of options that can be used with the configure script. To display these options, the following command is used:

```
config_office> ./configure --help
```

After running configure, run the source configuration file which sets all environment variables:

```
$SRC_ROOT> tcsh
$SRC_ROOT> source LinuxIntelEnv.Set
```

To create the build tools, the following command is used:

```
$SRC_ROOT> ./bootstrap
```

The bootstrap utility creates the tools required for building. This involves already building some modules and will take a few minutes.
Build Instructions

Building a Full Build of the Office Suite

At this point OpenOffice.org is ready for building. To build the entire suite, having created the environment as described above the dmake command is run from the top-level directory. This will take several hours.

$SRC_ROOT> dmake

If a need arises to rebuild a module or build each module individually (careful checking of dependencies is highly recommended!), the use of the build tool is necessary. A subsequent deliver will copy all created binaries, libraries etc. into the solver tree:

$SRC_ROOT/(module)> build

$SRC_ROOT/(module)> deliver

A note as to build time:

Using an Intel Architecture with a Pentium III processor, CPU speed 700 Mhz, supporting 256 MB RAM, uses 2 GB of disk space on an IDE drives and runs for a few minutes longer than 12 hours to complete.

Building Individual Projects with a Prebuild

OpenOffice.org is organised in several projects. For example, the Word Processing Project. These in turn consist of several modules, organised in separate directories. The source contains approximately 90 modules.

Users can build any project or module individually. Building modules individually
should not be misunderstood as reducing OpenOffice.org to a special application, say,
for instance, the spreadsheet application. The program will always consist of the
entire office suite: text processor, spreadsheet, drawing application, etc. Building
individual modules is useful when development work is being done on a certain
module. Most modules will depend on other modules having already been built. In
other words, all modules must build in a particular order. To avoid building all
modules which are prerequisites of the module of interest, users can make use of a
pre-build solver tree against which users can build any module.

In order to create the build environment and build tools (bootstrap), users also have to
check out the modules config_office, dmake, external, xml2cmp, mkdepend, sal,
solenv, vos, tools and rscpp.

To build a project, users build each of its modules individually in their directory with
the build tool, followed by deliver to copy the created libraries, binaries etc. into the
solver tree:

$SRC_ROOT/(module-name)> build
$SRC_ROOT/(module-name)> deliver

Files called build.lst in the directories (module-name)/prj contain all information
about the subdirectories to be build (each of them containing makefiles makefile.mk),
about internal dependencies, and also about modules the current module depends on.
The files (module-name)/prj/d.lst control the actions done by deliver. The last or
second to last directory to be built is usually module-name/util which is responsible
for linking one or more shared libraries.

Building a Project with Debug Information

To rebuild a complete project with debug information, remove all object files by removing the unxnghi3.pro directory. Then run build with the debug option set to true:

```bash
$SRC_ROOT/(module)> rm -rf unxnghi3.pro
$SRC_ROOT/(module)> build debug=true
```

Instructions to Build an Installation Set

The build process (started with a top-level dmake or build -all in $SRC_ROOT/instsetoo) will create installation sets in english and german. A simple build in $SRC_ROOT/instsetoo will also create the installation sets, provided all other modules are already built.

If users have build an installation set earlier and want to re-build it, users must delete the local outpath first:

```bash
$SRC_ROOT/instsetoo> rm -rf unxnghi3.pro
```

The installation set will be located at $SRC_ROOT/instsetoo/unxnghi3.pro/01/normal. Execute the setup binary to install: $SRC_ROOT> CD instsetoo/unxnghi3.pro/01/normal normal> ./setup

The 01 in the path names indicates that the localisation is American English. This number corresponds to the international phone code for the USA. The German installation set will be located in a subdirectory 49. This scheme holds true for all
localisations users may have chosen explicitly (see next section Building Localised Versions of OpenOffice.org).

**Building Localised Versions of OpenOffice.org**

Running the configure script with the `--with-lang` option will introduce the build of additional language resources. This option will introduce a command in the environment settings file which in turn after execution sets a variable like, for instance, `RES_FREN` to `TRUE` in the case of French (users can also set this variable by hand in order to introduce another language). It is also possible to build more than one language at once. (OpenOffice.org 2002)
Appendix II

Experimental Source Code

Sender.cc

/******************************************************************************

Sender PVM (master) program code by Sherine Antoun based on example code
(Fulcher 1998, Geist 1994, Deitel, 1998). Spawns slaves that copy files from
central location to local host, as each slave requires write privileges on local
drive, the master must run as UID of authorised account holder on local machine
and thus the slaves will inherit the same UID.
*********************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <fstream.h>
#include "pvmS.h"
#define DEBUG

main(int argc, char *argv[])
{
    int cc, tid, msgtag, bytecount;
    long filesize=0;
    char buff[100];
    char test3;
    char machine[1024];
    char lbuf[1024];
    char test[10][20];
    cc=tid=msgtag=0;
    sprintf(test[0], argv[0]);
    sprintf(test[1], argv[1]);
    sprintf(test[2], argv[2]);
    #ifdef DEBUG
    pvm_catchout(stdout);
    printf("i'm t\%x\n\n", pvm_mytid());
    cerr<<"Master contents of argument1 is :");<<test[0]<<endl;
    cerr<<"Contents of argument2 is: ");<<test[1]<<endl<<"Contents of argument 3 is
"<<test[2]<<endl<<endl;
    printf("i'm t\%x\n\n", pvm_mytid());
    #endif

    //having read 1st machine id from argument, fire slaves on 3 machines

    for(int v = 0; v < 3; v++)

S. M. Antoun

115
```c
{ int temp = atoi(argv[1]);

#ifdef DEBUG
    int rtemp = 01;
    cerr << "Value of machine name argument is: " << temp << endl << "Value of incremented rtemp is: " << rtemp + v << endl;
#endif

if ((temp + v) < 10) {
    sprintf(machine, "%s%d", "java", (temp + v));
} else {
    sprintf(machine, "%s%d", "java", (temp + v));
}
#endif DEBUG

cerr << "Value of machine argument in spawn call is: " << machine << endl;
#endif DEBUG

if (0 == (strcmp(machine, "java01"))) { cc = pvm_spawn("recipient", 0, 1, "java01", 1, &tid); }
if (0 == (strcmp(machine, "java02"))) { cc = pvm_spawn("recipient", 0, 1, "java02", 1, &tid); }
if (0 == (strcmp(machine, "java03"))) { cc = pvm_spawn("recipient", 0, 1, "java03", 1, &tid); }
if (0 == (strcmp(machine, "java04"))) { cc = pvm_spawn("recipient", 0, 1, "java04", 1, &tid); }
if (0 == (strcmp(machine, "java05"))) { cc = pvm_spawn("recipient", 0, 1, "java05", 1, &tid); }
if (0 == (strcmp(machine, "java06"))) { cc = pvm_spawn("recipient", 0, 1, "java06", 1, &tid); }
if (0 == (strcmp(machine, "java07"))) { cc = pvm_spawn("recipient", 0, 1, "java07", 1, &tid); }
if (0 == (strcmp(machine, "java08"))) { cc = pvm_spawn("recipient", 0, 1, "java08", 1, &tid); }
if (0 == (strcmp(machine, "java09"))) { cc = pvm_spawn("recipient", 0, 1, "java09", 1, &tid); }
if (0 == (strcmp(machine, "java10"))) { cc = pvm_spawn("recipient", 0, 1, "java10", 1, &tid); }
if (0 == (strcmp(machine, "java11"))) { cc = pvm_spawn("recipient", 0, 1, "java11", 1, &tid); }
if (0 == (strcmp(machine, "java12"))) { cc = pvm_spawn("recipient", 0, 1, "java12", 1, &tid); }
if (0 == (strcmp(machine, "java13"))) { cc = pvm_spawn("recipient", 0, 1, "java13", 1, &tid); }
if (0 == (strcmp(machine, "java14"))) { cc = pvm_spawn("recipient", 0, 1, "java14", 1, &tid); }
if (0 == (strcmp(machine, "java15"))) { cc = pvm_spawn("recipient", 0, 1, "java15", 1, &tid); }
if (0 == (strcmp(machine, "java16"))) { cc = pvm_spawn("recipient", 0, 1, "java16", 1, &tid); }
if (0 == (strcmp(machine, "java17"))) { cc = pvm_spawn("recipient", 0, 1, "java17", 1, &tid); }
if (0 == (strcmp(machine, "java18"))) { cc = pvm_spawn("recipient", 0, 1, "java18", 1, &tid); }
if (0 == (strcmp(machine, "java19"))) { cc = pvm_spawn("recipient", 0, 1, "java19", 1, &tid); }
if (0 == (strcmp(machine, "java20"))) { cc = pvm_spawn("recipient", 0, 1, "java20", 1, &tid); }
if (0 == (strcmp(machine, "java21"))) { cc = pvm_spawn("recipient", 0, 1, "java21", 1, &tid); }
if (0 == (strcmp(machine, "java22"))) { cc = pvm_spawn("recipient", 0, 1, "java22", 1, &tid); }
if (0 == (strcmp(machine, "java23"))) { cc = pvm_spawn("recipient", 0, 1, "java23", 1, &tid); }
if (0 == (strcmp(machine, "java24"))) { cc = pvm_spawn("recipient", 0, 1, "java24", 1, &tid); }
if (0 == (strcmp(machine, "java25"))) { cc = pvm_spawn("recipient", 0, 1, "java25", 1, &tid); }
if (0 == (strcmp(machine, "java26"))) { cc = pvm_spawn("recipient", 0, 1, "java26", 1, &tid); }
if (0 == (strcmp(machine, "java27"))) { cc = pvm_spawn("recipient", 0, 1, "java27", 1, &tid); }
if (0 == (strcmp(machine, "java28"))) { cc = pvm.spawn("recipient", 0, 1, "java28", 1, &tid); }
if (0 == (strcmp(machine, "java29"))) { cc = pvm.spawn("recipient", 0, 1, "java29", 1, &tid); }
if (0 == (strcmp(machine, "java30"))) { cc = pvm.spawn("recipient", 0, 1, "java30", 1, &tid); }
if (0 == (strcmp(machine, "java31"))) { cc = pvm.spawn("recipient", 0, 1, "java31", 1, &tid); }
}
}
```
cerr<<endl<<"Spawn fail.
 on machine number: "<<machine<<" ... fatal error for this machine 
"<<"skipping
.....return value was: "<<cc<<endl;
exit(1);
}
}
#endif DEBUG
#endif DEBUG

#ifdef DEBUG

cerr<<endl<<"Master the pvm_spawn returned  "<<cc<<endl<<endl;
#endif

pvm_exit();
exit(0);
}
Receive.cc

*********************************************
Receiver PVM (slave) program code by Sherine Antoun based on example code (Fulcher 1998, Geist 1994, Deitel, 1998). Copies files from central location to local host, requires write privileges on local drive, must run as UID of authorised account holder on local machine

*********************************************

#include <fstream.h>
#include <unistd.h>
#include <iostream.h>
#include "pvm3.h"
#define DEBUG

int main(int argc, char*argv[]) {

int ptid, cc, bytecount;
char * machinename= new char[1024]; //should be machine name
char * lbuf= new char[1024];

long filesize=0, cursize=0;
long*myfile=new long;
int msgtag =1;
char sourcepath[1024];
char destpath[1024];
char command[1024];

memset(lbuf, '\0', 1024);
ptid = pvm_parent();

//open machine name file
ifstream infile("machine.dat", ios::in);
if (!infile) {
#endif DEBUG
cerr<<"Error 01. Machine cannot be identified, missing machine.dat file,"<<endl<<"archiving to unkown machine aborted" <<endl;
#endif exit(l);
}

//read machine name

S. M. Antoun  118  Appendix ii
infile>>machinename;

//copy relevant directory to /spare0/sma02

sprintf(command, "cp -r /packages/tmp/%s /spare0/sma02", machinename);

#ifdef DEBUG
    cerr<<"The system call is: "<<command<<endl;
#endif

system(command);

//write updated log on /packages/tmp

//change to archive directory
chdir("/spare0/sma02");

//send date to logfile
sprintf(command, "date >>/packages/tmp/%sfiles.log", machinename);
system(command);

//append to file current directory listing
sprintf(command, "ls >>/packages/tmp/%sfiles.log", machinename);
system(command);

pvm_exit();
exit(0);
Makefile

*******************************************************************************

Makefile for Recipient and Sender PVM (master / slave) program code by Sherine
Antoun based on example code (Fulcher 1998, Geist 1994, Deitel, 1998, Ward
2001). Compiles against PVM library, this make file will only works within the
SITACS network as the libraries are referred to by absolute paths:
"/packages/pvm3/lib/LNIX/ -lpvm3 -lsocket -lnsl" and
"/usr/share/pvm3/include".
*******************************************************************************

SHELL=/bin/sh

CC=g++

INCLUDES=-I/usr/share/pvm3/include

LIBS=-L/packages/pvm3/lib/LINUX/ -lpvm3 -lsocket -lnsl

CFLAGS=-g $(INCLUDES)

.SUFFIXES: .cpp $(SUFFIXES)

.cpp.o:
   $(CC) $(CFLAGS) -c $<

.cc.o:
   $(CC) $(CFLAGS) -c $<

TARGETS=sender recipient

All: $(TARGETS)
   cp $(TARGETS) $(HOME)/pvm3/bin/SUN4SOL2/.

Clean:
   rm -f *.o $(TARGETS)

#sender.o
#recipient.o

sender: sender.o
   $(CC) -o $@ sender.o $(LIBS)

recipient: recipient.o
   $(CC) -o $@ recipient.o $(LIBS)
Corruptafile

Corruptafile program code by Sherine Antoun based on example code (Deitel, 1998). Interactive program prompts users for original archival files and rate of noise to be artificially introduced, restricted to the experimental levels 3, 7, 11 and 18%, and writes the modified files out to "c_xxx" where xxx is the original file name, the process is repeated at the same artificial noise level until "@" is input as archival file name.

```cpp
#include<iomanip.h>
#include <iostream, h>
#include <fstream.h>
#include <stdlib.h>
#include <string.h>
#define DEBUG

int main() {
    char a;
    int count=1, b=0;
    char Ibufin[1024];
    char Ibufout[1024];
    memset(Ibufin, '0', 1024);
    memset(Ibufout, '0', 1024);
    while(1)
    {
        system("clear");
        cout<<"File name for noise introduction or @ to terminate: ";
        cin >> Ibufin;
        ifstream fin(Ibufin, ios::in);
        sprintf(Ibufout, "c_%s", Ibufin);
        ofstream fout(Ibufout);
        if (!fin)cout<<endl<<"Error input file open FAILED check file and try again."
        "<<endl<<endl, exit(1);
        do{
            cout<<endl<<"Artificial noise level amount?"<<endl;
            cout<<"For 3% noise 33, for 7% noise 14, for 11% noise 9, and for 18% noise 6"<<endl;
        cin>>()b;
        if (b == 33)continue;
        if (b == 14)continue;
        if (b == 9)continue;
        if (b == 6)continue;
        b=0;
    }
}
```

S. M. Antoun 121 Appendix ii
} while (b == 0);

while (fin.good()) {
    fin >> noskipws >> a;
    #ifdef DEBUG
        cout << noskipws << a << endl << endl;
        cout << "Corupted" << endl << endl;
        cout << (a) * (-1) << endl;
    #endif
    if (!(!count % b)) {
        fout << (char) a * (-1);
    } else {
        fout << noskipws << a;
    }
    count++;
    if (count == 30000) count = 1;
}
#endif
    cout << endl << "the count is: " << count << endl;
#endif
}
return 0;
Appendix III Publications from this Thesis

"An Open Source Approach to Medium-Term Data Archiving"

Sherine Antoun, John Fulcher & Carole Alcock
School of IT & Computer Science
University of Wollongong
Australia.

Abstract

Medium- to long-term archiving of digital documents – beyond the lifespan of the authoring software/hardware – is quite a challenging problem. Magnetic and optical media are susceptible to environmental influences, and deteriorate over time, often to the point where the archived documents can no longer be retrieved. Previous attempts to address this problem include migration and emulation, both of which have their attendant difficulties. It is the contention of the present study that an Open Source approach offers several advantages. More specifically, by archiving the Open Source application programs (in source code, not executable form) along with the documents in question – in both plain and compressed form – then this significantly increases the likelihood of being able to retrieve such archives at some future time. The application source code can be re-compiled to a form suitable for reading in (Open Source) viewers, thereby presenting to the user the archived document as the original author envisaged it. One set of experiments was undertaken distributing documents together with their (Open Source)
authoring software via a Portable Virtual Machine (PVM) program to unused disk space on a network of SUN workstations. The success of this approach was evaluated using the following four measures (i) lossiness of conversion, (ii) edit-ability, (iii) ability to save back to the original format, and (iv) functionality retention. Another series of experiments was conducted in which artificial ('speckle' or salt-and-pepper) noise was deliberately introduced to the archived documents in order to mimic degradation of the storage medium over time. It was found that survivability was heavily dependent on file type: simple text files and MPEG movies were impervious to even 18% introduced noise. Source code programs and JPEG images, by contrast, were intolerant to even the smallest noise levels (it has to be said however that straightforward re-editing of the former led to error-free compilation without much difficulty). Lastly, it was found that decompression (specifically the publicly available RAR decompressor) further enhanced the file recovery process. We conclude that an Open Source approach to the preservation of digital archives has considerable potential.

1. Introduction

Manuscripts from antiquity are still capable of being read in this day and age, due in large part to the medium on which they were originally created – "etched in stone", as it were. Preservation of records from more recent eras has proved more problematic: papyrus/paper records are susceptible to fire, flood and silverfish; early (acetate?) film stock was quite volatile – indeed, much was recycled for its silver content – and magnetic media decays over time (optical media less so). In other words, the more modern media
are a lot more fragile compared with granite. Archival storage of digital records is even more difficult, since bit loss can render the entire document unreadable. As Kahle (1996) rightly observes: "while it is possible to read 400 year old books printed by Gutenberg, it is often difficult to read a 15 year old disk". To cite another example: while the original lunar landing capsule (artefact) lives on in a NASA museum, its controlling software no longer exists (Open Channel Software, 2004).

Coupled with epoch shifts of media types has been the increase in mass availability of documents. At no other time has this been more the case than in this information age. Other characteristics of digital media are its ever-increasing capacity, its changing formats and the obsolete nature of authoring software (Rothenberg, 1999). All of this renders the task of archiving today's documents for prosperity all the more challenging.

The past decade has witnessed considerable activity in attempts at archival longevity (Laurie, 2000; Ross, 1999; Rothenberg, 1999, 2000, 2001; Waugh, 2001). Common approaches have been data migration, emulation and archival of the original authoring software along with the archival documents themselves, all with limited success. Simply stated, long-term digital archiving is the storage of unstable bit patterns on unstable media, for periods of time vastly exceeding the expected media lifespan, with subsequent (lossless) retrieval on obsolete equipment – quite a challenge!

Given the rapidly changing nature of digital records, can we devise a better approach to medium- to long-term archiving? It is the contention of the present study that the use of
Open Source software can go part of the way to achieving this aim.

2. Digital Archiving Methods

The survival of a digital document is the ability to retrieve an archive, and for it to appear as the author saw it at the time of creation. It encompasses not only physical, but also functional and content survival. Beagrie (2001) defines long-term preservation as encompassing two distinct but equally important functions: (i) long-term maintenance of a byte stream, and (ii) on-going access to its content through time and changing technology. Furthermore, the fundamental principle behind any digital preservation strategy must be to ‘do minimal harm’ (Waugh, 2000).

One approach to preservation is migration, defined as the periodic transfer of digital material from one hardware/software platform to another, or from one generation of computer technology to a subsequent generation (Muir, 2001). It has been used for decades to preserve operational data in data processing installations (Lorie, 2001). Another approach is emulation.

One difficulty with migration is that it requires a conscious act on the part of users. Time and financial constraints often cause the deferral of such tasks to a point in time when the old records deteriorate and become irretrievably lost (Rothenberg, 1999). The migration process proper involves physical changes to the stored data from the older format to the newer (involving binary representation, layout, encoding). The cumulative effect of
repeated changes can however result in degradation of the evidentiary status and/or value of the digital document (Thibodaux, 2001). Further, the migration (conversion) process itself can potentially degrade or corrupt the digital document (Rothenberg 2000; Waugh 2000; Henrard 2002).

With emulation, one (leading edge) computer system reproduces the behaviour of another (trailing edge) system. In this manner the emulator can run software designed for the (obsolete) system it is emulating (Rothenberg, 2000). Emulation can therefore be regarded as preservation based on technological advancement. Of necessity, it requires extensive knowledge of obsolete hardware architectures, operating environments and application software in order to build a virtual machine on a modern system environment.

Lorie (2001) developed a prototype in order to demonstrate the long-term viability of emulation for digital preservation, based on a "Universal Virtual Computer". This UVC approach stores the technical specifications for a simple, software-defined decoding machine in paper form, which potentially at least could last for centuries (silverfish, flood and fire notwithstanding). The key word here is 'simple' – it remains to be seen whether such detailed specifications can indeed be distilled into a brief paper document. Rothenberg (2000) developed an emulation system which stored the original (proprietary) authoring software along with the digital documents and a detailed human-readable description of the architectural environment of the current machine (in order to facilitate the development of emulators by future users).
Other, more esoteric, approaches have been suggested, including buckets (Nelson, 2001), VERS (Waugh, 2000) and genomic storage (Wong, 2003).

3. Why Open Source?

Are migration, emulation or other techniques capable of delivering medium- to long-term archival storage? Let us consider each in turn.

Firstly, after several successive migrations (or refreshing and conversions), a document may bear little resemblance to the original (Muir, 2001). Worse, it may be impossible to subsequently determine just what has been lost (Waugh, 2000). Many organizations manage their information poorly, with the result that the migration process misses significant amounts of information (Waugh, 2000). Also, there is often a temptation to delay migration too long, past the point where information can be converted in a cost effective manner. Even if migration is timely and diligent, it can be less than effective because the target format is not 100% compatible with the source format (for example, a spurious comma in a comma-delimited spreadsheet could throw the entire document into disarray).

If migration is unsatisfactory, then perhaps adapting modern equipment to run old software (i.e. emulation) is. We face a new set of challenges with this approach. Invariably, software is Operating System- (and often hardware-)specific (Muir, 2001). Our options therefore boil down to (i) OS preservation (ii) hardware preservation (iii) OS...
simulation or (iv) machine simulation. Unfortunately the latter approach is often only a “best guess” effort, in the absence of museum machines for comparison purposes. Indeed, Bearman (1999) ponders whether preserving the OS and/or machine is preserving the wrong thing; in other words, is the target of preservation the system or the document?

The emulators of Lorie and Rothenberg both required storage of detailed system specifications along with the archival document(s). As systems become ever larger and more sophisticated, this becomes more and more of a challenge, and amounts to an exercise in complexity management. As Franz (1993) has observed though, developers need to overcome such limitations as:

- incompatible paradigms (e.g. object-oriented versus procedural),
- contrasting abstractions (e.g. Oberon differentiates between a data file and its access mechanism; Macintosh doesn’t), and
- the target system may have no equivalent constructs (e.g. a limit on the number of concurrently open files).

Buckets are storage units that contain data and metadata, as well as methods for accessing both. They are aggregative, intelligent, internet-accessible digital objects optimized for publication in Digital Libraries, and follow a Smart Object, Dumb Archive model. The tightly integrated nature of buckets severely limits their tolerance to change. For example, system changes require pointers to application and document locations within the bucket construct to be updated. Over time, this collection of pointers can become entangled and the risk of loss escalates. Failure to maintain the system can lead to data loss, context...
loss, reference loss and eventual loss of the entire document.

VERS objects encapsulate the information to be preserved within metadata that describes aspects of this information, such as data formats and descriptions of the preserved information (Waugh, 2000). In short, VERS is designed to be self documenting and self contained. From the perspective of digital archiving though, VERS is limited because it constrains users to a specific set of (proprietary) application software, running on a (proprietary) OS.

The need for a wet laboratory and the as yet unproven reliability of genomic storage (for example, random mutations in and non-survival of the host bacteria organism) renders it an unlikely candidate for long-term archival storage.

We thus conclude that all the above approaches to digital archiving are severely flawed. What then can we propose as an alternative? Previous efforts have tended to seek a single, unique solution (i.e. panacea) – likewise they tend to rely heavily on periodic user intervention. Not surprisingly, the goal has proved elusive. The approach taken in this study revolves around the use of Open Source software, which we believe has not previously been used in the context of medium- to long-term digital archiving. In doing so, we circumvent many of the problems previously outlined. Further, we adopt the Future Digital Archiving philosophy of facilitating the recovery effort where preservation either fails or is likely to fail.
Proprietary software is often expensive, requires periodic upgrading and is only available in executable, not (ASCII text) source code form. By contrast, Open Source software is free and the source code readily available. The advantages of Open Source can be summarized as follows (Open Source Initiative – OSI: http://www.opensource.org/licenses/ last accessed March 19 2004):

the ability to view and edit source,
the ability to port the application from one platform to another,
the ability to modify the software functionality to suit specific user requirements, and
the ability to store, copy and install applications at will without a licensing fee.

Under the General Public License (GPL), ported source code must also be published as Open Source; modified source code may or may not be published as Open Source, but may not be rendered proprietary. The flexibility afforded by the GPL licensing system, coupled with the availability of application source code, renders Open Source software uniquely suited to medium- to long-term archival. The ability to edit/modify source gives it the longevity needed for the survival of digital documents beyond the current technology era.

The rationale behind storing source code along with the document is consistent with the FDA premise of being the ‘contingency of last resort’, namely where there is the potential for future users to modify the source in order to render the archived document(s) compatible with contemporary systems.

The aim of this study was to assess the viability of Open Source software as a robust,
low-cost, low-overhead and widely available method for medium- to long-term digital document preservation, capable of transcending system and software obsolescence, while delivering to future users the documents as their authors saw them.

4. Experimental Results

In order to validate our Open Source approach to medium- to long-term digital archiving, we undertook a series of experiments. These involved (a) fetching and installing the relevant Open Source programs, (b) archiving documents on various media, (c) restoring the archived files, and (d) assessing the passage of time on the archived documents. All media deteriorate over time. For example, magnetic media is susceptible to temperature, humidity, and magnetic fields; Optical media can suffer from both oxidisation of the internal reflective material and from fungal infection.

4.1 Open Source Application Software

The primary experimental platform was an 800MHz Dell Latitude LS400 laptop computer running (Open Source) Mandrake V9.0 Linux (public release 2002). The Open Source applications – all obtained from http://sourceforge.net – used in this study were:

- Open Office (desktop publishing)
- Xpdf (PDF reader)
- Xine (video/DVD player)
- The Gimp (graphical editor)
- Galleon (web browser)
These representative Open Source tools were all compatible with their proprietary equivalents, to a greater or lesser extent. The application which exhibited most variation from its proprietary counterpart was Open Office. Compared with MicroSoft's Office-XP suite, we experienced mixed results: MS-Word, Excel and Powerpoint documents could all be opened, viewed and manipulated within Open Office, with most of their formatting and functions intact, but MS-Word macros proved incompatible. In the opposite direction, 'swx' files could be readily exported to MS Office (.doc) from Open Office. The online help within Open Office is somewhat incomplete, terse and lacks context, compared with the step-by-step online help provided in MS Office. Open Office—unlike MS Office—does not offer integrated email or a calendar accessory. Lastly, Open Office does not support collaborative editing tools. A comparison of both Office suites is given in Table 1.
<table>
<thead>
<tr>
<th></th>
<th>Open Office</th>
<th>MS Office</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format compatibility</td>
<td>Supports .swx, .doc &amp; .rtf</td>
<td>Supports .doc &amp; .rtf</td>
</tr>
<tr>
<td>Graphics handling</td>
<td>Lower quality; able to display MS Office graphics, but lower quality &amp; no 3D graphics ability</td>
<td>High quality, crisp, sharp, well-textured graphics &amp; 3D</td>
</tr>
<tr>
<td>Help files</td>
<td>Terse, incomplete, no context &amp; no step-by-step instructions</td>
<td>Complete, good context &amp; step-by-step instructions</td>
</tr>
<tr>
<td>Integration</td>
<td>Poor; often need to be manually started; no collaboration support</td>
<td>Seamless integration of email &amp; calendar; collaborative work supported</td>
</tr>
<tr>
<td>Disk usage (full install)</td>
<td>275MB</td>
<td>233MB</td>
</tr>
<tr>
<td>OS’s supported</td>
<td>WIN98, 98SE, 2000, NT, XP; SolarisV7.8; Linux Kernel 2.2.13 or higher &amp; glibcV2.1.3 or higher</td>
<td>WIN98, 98SE, NT4.0 with SP6 only, XP</td>
</tr>
</tbody>
</table>

Table 1 – Features of Open Office and MS Office.
4.2 Archiving of Documents together with their Authoring Software

The collection of Open Source application software described in Sect. 4.1 was stored on a variety of media, these being:

(fixed) magnetic disks (hard drives),

(removable) magnetic disks (both floppy and Zip disks)

(removable) optical disks (CD-R)

In the case of the fixed magnetic medium, a multi-threaded PVM (Parallel Virtual Machine) program – Archiver – was developed in order to distribute the digital archive to free disk space on the network of SUN Ultra Workstations shown in Figure 1.
All 36 network nodes are housed within a single laboratory within the School of Information Technology & Computer Science at the University of Wollongong. They connect to a CISCO switch by way of 10/100 twisted pair Ethernet, thence by 1Gigabit fibre optic cable to the building switch which is connected to the main University server. Each SUN Ultra node (workstation) boasts 30GB of unused disk space, since account users are allocated storage by NFS on a remote RAID. Thus the total available (distributed) archival storage space available for this study was 1.08TB. From the perspective of this study, these 36 SUN workstations constituted a form of RAID.
Archiver supported the management, transport and access of archived documents over the network, and was based on a design used in climate modelling and other fields where large data sets need to be accessed over networks (Allcock, 2001). Multi-threading led to improved performance when transporting large amounts over data over the network. Apart from storing the digital archives per se, fragmented, redundant, compressed copies were also stored on the network disks (generated using the RAR compression tool [http://www.win-rar.com/]). While the RAR compression software is proprietary, the RAR decompressor is Open Source, and thus falls within the scope of this study.

Complete archives were stored on both the (fixed) networked and (removable) optical disks (which took three 700MB CD-Rs). A subset was archived to a 100MB Zip disk, and an even smaller subset to a 1.4MB 3.5" floppy disk. These archives comprised the following:

- personal documents accumulated over a 6-year period,
- files obtained from the world wide web (mainly electronic libraries, where permission is granted for personal/educational use),
- files available under public license, and
- promotional material publicly available on the internet.

During the experiments, a (simple ASCII text) central catalogue of the archived files was maintained, along with the network node on which each file resided as well as the appropriate viewing tool. Whenever the archive was modified this central catalogue was
updated and copied to all network nodes.

Archive retrieval was document-based, with appropriate (Open Source) viewing tools being used to access the desired documents. Apart from accessing the archived documents in their raw form, the RAR decompressor was also used to retrieve the redundant, compressed archives also stored on the network. This decompression tool provided error recovery/correction, which is an essential aspect of any archiving effort, in order to maintain the viability of the archive. Using multiple, redundant copies can also provide a recovery mechanism, since future users would be able to mix and match fragments from the multiple copies in order to reconstruct an error free archive.

The following four measures were developed in order to both evaluate the retrieved archive and to assess the effect of noise:

- lossiness of conversion (where applicable),
- ability to edit the data files (except for PDF),
- ability to save back to the original format, and
- functionality retention (e.g. embedded formulae, macros, hyperlinks etc.)

A Boolean measure – ‘true’ or ‘false’ – was gathered for each applicable criterion. For example, if a file could be successfully opened without generating an error message, then ‘no-lossiness’ was marked as ‘true’. Results obtained at various levels of introduced noise are discussed in the following Section.

4.3 Simulating the Passage of Time via Introduced Noise
In this series of experiments, we deliberately introduced varying degrees of noise into the archived files in order to mimic deterioration of the storage medium over time. This was based on the premise that increasing damage eventually leads to a file becoming unusable.

A program was written which opened each file in the archive, read its contents, added a predetermined amount of noise, then wrote the result to a modified file. Four levels of noise were tried – 3%, 7%, 11% and 18%. The lower levels simulated distributed noise caused by adverse storage conditions; higher noise levels corresponded more to severe damage, such as that resulting from mechanical (physical) damage to the storage medium (for example, scratches on the surface of a CD are contiguous rather than uniform/random). The noise manifested as inverting the bits in every thirty-third character (3%), every 14th character (7%), and so on. In the experiments conducted during the course of this study, any noise distribution could have been synthesized; we actually opted for salt-and-pepper (speckle) noise, since this corresponds to data dropout in real-world systems (and leads to 'salt-and-pepper' like spikes).

Archives comprised the following file types:

MS-Word (.doc)
Documents with embedded Cyrillic characters
Documents with embedded graphics (.wmf; .gif)
Documents with embedded images (.bmp; .gif; .jpg)
Documents with embedded hyperlinks
Documents with embedded tables

Text only documents

MS-Excel spreadsheets incorporating csv formulas

Portable Document Format files, generated by a variety of applications (.pdf)

MPEG4 video files (.mpg)

QuickTime movie files (.mov)

JPEG image files (.jpg)

Audio files (various formats – e.g. .wav; MP3; WMA; Ogg Vorbis)

Program source code (C; C++; Perl)

In general, and not surprisingly, we found that the higher the proportion of introduced noise, the more adverse the effect on the stored archive.

Apart from source files, all document types displayed resilience (tolerance) to 3% added noise. All files were successfully opened using appropriate Open Source viewing tools. One file only out of 13 audio files failed to play back. All source code files failed to compile (needless to say, even a single bit flipped in an executable file renders it unusable – no such binary files were archived during this study, only source code). The results are summarized in Figure 2, in which the four criteria listed in Sect.4.2 are shown for each file type.
Figure 2 – Effects of 3% Introduced Noise

Increasing the level of artificially introduced noise to 7% resulted in 5 out of 456 .doc files exhibiting discernible errors. Likewise, 1 out of 20 spreadsheets, 1 in 20 PDF, and over half the JPEG images (12 out of 20) all contained errors. 3 audio files failed, and all source code files produced compile errors. All files opened successfully using the appropriate Open Source viewing tools however.

11% introduced noise produced errors in 56 .doc files, 5 spreadsheets, 7 PDF files and 15 JPEGs. 13 audio files failed (2 of 4 .mov trailers failed to run), and once again all source code files produced compile errors.

The highest noise level – 18% – yielded errors in 105 .doc files, and a further 20 files failed to open altogether. All 20 spreadsheets and 20 PDF files contained errors, with 17 spreadsheets and 8 PDFs failing to open. 7 JPEGs contained visible errors, with 13 failing to open ("this is not a jpeg file"), 4 .mov movie trailers failed, as did 13 audio files. As
previously, all source code files caused compile errors. These findings are summarized in Figure 3.

![Figure 3 - Effects of 18% Introduced Noise](image)

These results reveal a correlation between the amount of introduced noise and the proportion of adversely affected files, except in the case of source code files, where all files were affected (which when one thinks about it is to be expected). At some point – between 10% and 20% – most files become unusable. Even at the highest noise levels though, some file types remained useable – specifically simple text files and MPEG videos (although as regards the latter, some noisy frames in a 25 frame per second film will be undiscernible to the human eye, whereas others will be). More complex file formats, such as machine readable source files (especially those containing embedded binary executable routines) and spreadsheet files (with multiple worksheets and embedded formulas) were much more susceptible to noise, and hence prone to failure.

4.4 Aiding File Restoration by way of Decompression
Another set of experiments was undertaken to verify that backup compressed archives could assist in the recovery process. Indeed they did – the entire archive was able to be successfully rebuilt from compressed volumes. We therefore advocate the use of compression/decompression in any practical archival system, not with the aim of saving space but for the increased error tolerance that results. A recovery record contains up to 524,288 recovery sectors, each recovery sector being able to recover 512 bytes of damaged data. Generally speaking, the size of recovery record may be approximated using the following Equation 1:

\[ \frac{(\text{archive size})}{256} + \text{number of recovery sectors} \times 512 \text{ bytes} \]

*Equation 1.*

RAR has documented tolerance of up to 10% errors, or one lost volume in 10 of a fragmented archive (http://www.win-rar.com/).

5. Discussion

Based on our experimental results, we conclude that file format complexity and specific file use significantly reduce noise tolerance. The file types which were most impervious to introduced noise were characterized by:

Simple encoding (e.g. simple text documents), and/or

High levels of compression (coupled with approximation in the decompression algorithms).

Source code files – being machine readable ASCII text files – are very susceptible to
introduced noise, leading to compile time errors; they nevertheless remained *humanly* readable. With both ASCII text files and MPEGs, the viewer is quite tolerant of errors. A programmer skilled in the art of debugging could correct source code errors to the point where the file once again complies. Since MPEG decoding utilizes approximation in decompressing image data, this means that if one or more frames are compromised the human eye will not discern this (short of losing the file headers which contain the encoding and frame rate). Analysis of the movie frame-by-frame *would* reveal the compromised images though.

6. Conclusion

This study has demonstrated the fragility of digital documents one may typically require to archive. It has also confirmed the viability of using Open Source tools to retrieve files created by proprietary software. The one requirement of our approach is that the IT person responsible for archiving needs to be not only a capable programmer, but au fait with Open Source methods – in other words, a proficient Systems Administrator (but surely such a responsibility usually falls within a SysAdmin’s job specification in any case?).

While beyond the scope of the present study, we nevertheless recommend a longitudinal study be undertaken to further verify the validity of our Open Source approach over a longer time frame.
Bibliography


Permanent Records in Electronic Systems (InterPARES), pp.3-14.


FYI

Begin forwarded message:

From: "Ahmed Gomaa" <ahgomaa@pegasus.rutgers.edu>
Date: Thu Jul 8, 2004 3:19:17 PM Australia/Sydney
To: "John Fulcher" <john@uow.edu.au>
Cc: <jd@dljournal.org>
Subject: RE: Paper Submission

Dear Dr. Fulcher:

Thank you for your submission. Your paper log is 1166. For future submission, please submit your paper onto our web site at www.dljournal.org. The paper review takes between 3 and 4 month.

Kind regards
Ahmed

Ahmed Gomaa
PhD Candidate, Rutgers University
MSIS department - IT major
Research Associate, CIMIC
Assistant to Editors-in-chief, Journal of Digital Libraries
http://www.dljournal.org
email: ahgomaa@cimic.rutgers.edu, http://cimic.rutgers.edu/~ahgomaa

-----Original Message-----
From: John Fulcher [mailto:john@uow.edu.au]
Sent: Wednesday, June 30, 2004 11:40 PM
To: adam@adam.rutgers.edu
Cc: neuhold@ipsi.fhg.de; furuta@cs.tamu.edu; frommholz@ipsi.fraunhofer.de; ahgomaa@cimic3.rutgers.edu; John Fulcher; Sherine; Carole ALCOCK
Subject: Paper Submission

Dear Editors-in-Chief,

Being unfamiliar with your submission procedures, as an initial attempt I have attached a copy of a paper draft entitled "An Open Source Approach to Medium-term Digital Archiving" - S. Antoun, J. Fulcher & C. Alcock for consideration for publication in Intl. J. on Digital Libraries. Will this electronic copy suffice, or do I need to submit hard copies (& if so where to & how many copies)?

best regards,
Prof. John Fulcher,
University of Wollongong,
Australia.