Vrije Universiteit Amsterdam   Universiteit van Amsterdam

Master Thesis

Scoops and Brushes for Software Archaeology: Metadata Dating

Author: Robert Jansma (2507994/11159723)

1st supervisor: Gerard Alberts
2nd reader: Natalia Silvis

A thesis submitted in fulfillment of the requirements for the joint UvA-VU Master of Science degree in Computer Science

June 9, 2020

"Computer science is no more about computers than astronomy is about telescopes."

- Edsger Dijkstra

Abstract

Software archaeology is the field handling the recovery, preservation and study of digital material. Web archaeology is a subcategory of software archaeology focused on the Internet. The study of digital artefacts requires extensive knowledge of computer systems. The present research aims to lower the technical barrier for the study of digital artefacts. There is a lack of tools which do not require extensive system and data science knowledge to operate. In this thesis, the historical value of extracting time-related metadata from digital artefacts, metadata dating, is discussed. The extraction of the four time-related metadata, birth time, access time, modify time and change time, and how these can be used for historical analysis is demonstrated. A model for looking at the preservation of digital artefacts is proposed demonstrating the value and place of time-related metadata. A tool for automatic extraction of time-related metadata was developed for this purpose called the MetadataDating tool. Metadata dating is performed on three archives related to De Digitale Stad, a pioneering website of the early Dutch web. By meaningful aggregation of the time-related metadata, new historical insights have been gained into De Digitale Stad. The results demonstrate that the overall development of De Digitale Stad remained constant over its lifespan. Users of De Digitale Stad valued other types of files over code files, a preference that increased at the end of De Digitale Stad. The importance of good copies of digital artefacts is demonstrated by the inherent paradox that performing research can alter the time-related metadata of a digital artefact.

Keywords— Software Archaeology, History, Tools, Methodologies, Time-related Metadata

Contents

List of Figures

List of Tables

1 Introduction

Software archaeology is the field handling the recovery, preservation and study of digital material. The rise of the digital storage technology opens great opportunities for the preservation of our cultural heritage. Many lessons can be learned from classical archaeology, but digital artefacts come with their own opportunities and challenges. A digital artefact can be a single file, a folder, a collection of files and folders, or even an entire system, possibly comprising several devices.

With digital artefacts come developers, the builders of the artefact. Such historical actors may provide a valuable context in the form of oral histories and tacit system knowledge of the artefact.

Digital material also has its users, people who have used or experienced the artefact. They can provide a different perspective on the artefact compared to the developers.

Digital artefacts are not just lines of code or files in isolation. They come to life only in a technical and social context allowing them to be properly experienced [Teszelszky, 2019]. A website requires a browser to be viewed. Without, it would lose its functionality and experience. While modern files and programs might work on most modern computers, legacy systems often do not have the luxury of immediately working on modern systems. They require software and sometimes even hardware that is no longer present or available.

The context is not only a technical one but also a cultural one, including digital artefacts of the same type and a broader cultural understanding of the place of the artefact. On studying websites Niels Brügger describes the broader context as the web sphere wherein the website is placed and the larger World Wide Web as a whole [Brügger, 2009].

1.1 Questions for digital artefacts

For questions in the field of software archaeology, the best source for answers might well be the primary source, the running system itself. One might have access to the original hardware with the files still intact. Alternatively, system backups may have been made during development or use. These can function as powerfull sources of ground truths.

When presented with a storage device that contains the digital artefact one wishes to study, the obvious thing to do is to plug it in and start looking around. Plugging it in and looking around is where knowledge, or the lack thereof, starts to become a problem. On opening up the storage device one may see all sorts of things: a single file, a folder, several folders containing more folders, or even a complete operating systems.

In some cases, the archaeologist recognises the artefact or some of its components. Most people would recognise a JPG file as a picture. A folder containing office files would already require some more knowledge: for example, the office software used to create it. The more complex and rare the files, the more knowledge is required to know not only what one is dealing with, but also how to meaningfully perform research on it. The type of data determines which questions are logical to ask of it. For example, one may ask for the size of a file in many different ways. For any file, it makes sense to talk about its use of disk space in bytes. For code, it makes sense to talk about the number of lines, but such a metric would make little sense for pictures. JPG files are not organised by line, so pixel dimensions would be far more appropriate.

When dealing with more complex software systems, consisting of multiple files and file types, more in-depth knowledge will be required to pose and to answer the relevant questions. General knowledge of how the system operates is essential, often including tacit knowledge, which is only present with people who have worked with comparable systems or even only in the minds of the developers themselves [Alberts et al., 2017]. For example, what is the typical location and structure of the folder(s) to manage external projects? Which could be done anywhere, whereas different groups use different locations. The more access to a system relies on tacit knowledge, the less use general tools will have. In cases where system-specific knowledge is needed, tools will have to be either adapted or rewritten on a per-system basis.

1.2 Metaphors and exploring the early web

The metaphor of archaeology is used to describe the historical work performed on digital artefacts. The archaeology metaphor helps to understand what a digital archaeologist does. They study artefacts much like traditional archaeologists [Teszelszky, 2019].

The scoops and brushes metaphor is used to explain the use of software tools in digital archaeology. These are used to unearth digital artefacts and form a vital part of the study of digital artefacts.

The use of metaphors in digital spaces used to be very common in the early web to help understand the technology. In De Digitale Stad (DDS), seen in Figure 1.1, the metaphor of a city was used to help explain the functionality of the website [Rommes et al., 1999].

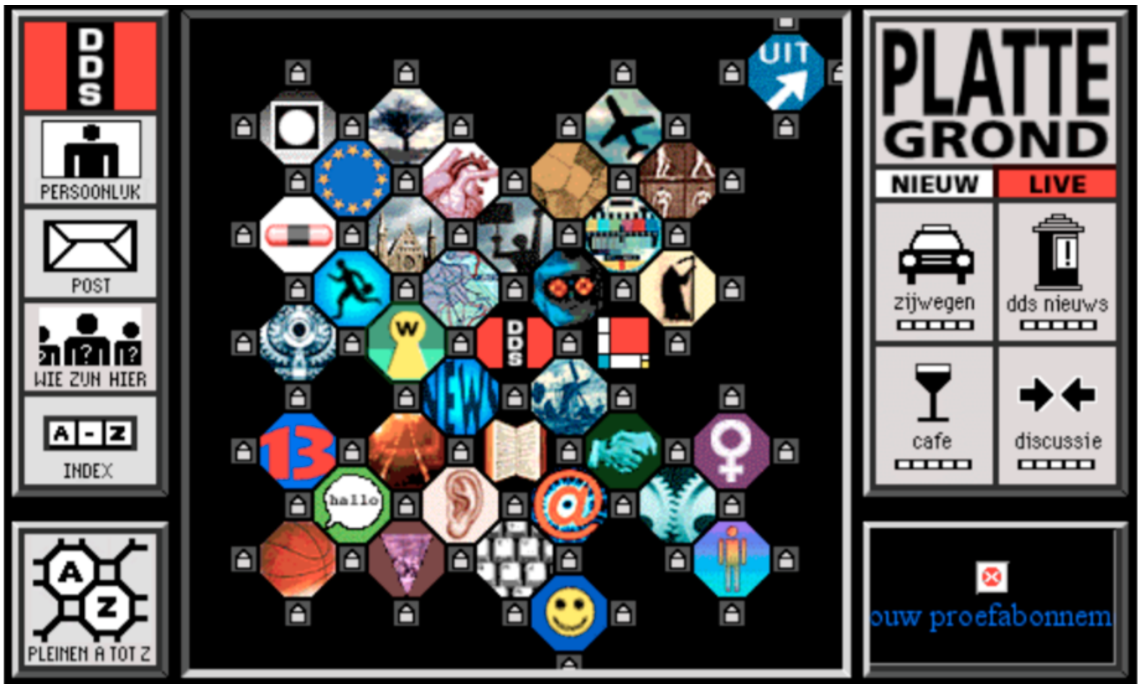


Figure 1.1: DDS interface

De Digitale Stad was one of the websites defining the early Internet in the Netherlands. Launched in 1994, it lowered the threshold for the general public to get online. DDS was initially funded by the municipality of Amsterdam as a ten week experiment with the goal to decrease the gap between politicians and citizens through the new medium of the internet. Internally, the goals of DDS focused more on experimenting with the new technology. Although the goal of political engagement was not achieved, DDS was massively popular and it was decided that DDS would be institutionalized and carried on [Rustema, 2001].

DDS has played an important role in the development of the digital culture in the Netherlands. DDS made the internet more accessible to the general public and provided a public space for programmers, hackers and artists to express themselves [Riemens and Lovink, 2002].

On its 2nd anniversary in 1996, DDS made a backup of all servers hosting the website. The backup was called the FREEZE. It encompassed three servers named dds, shaman and alibaba.

In the late 90's DDS lost its early bird advantage on the web and started to suffer from competition, resulting in its commercialization in 2000. DDS ended its free services on the first of September 2001 [DDS, 2001].

The Amsterdam Museum is the museum dedicated to the history of Amsterdam. In 2011 the Amsterdam museum started the RE:DDS project, intending to collect material related to DDS and to reconstruct the website [De Haan, 2016d]. On 13 May 2011, Waag Society hosted the Grave Diggers Party [Waag, 2014], an event with the goal to collect material related to DDS. People who had been either a member or otherwise involved with DDS were invited to bring their old computers, floppy drives, servers or any other materials related to DDS, also including non-digital materials such as manuals, posters and stories [De Haan, 2014b].

Shortly after the Grave Diggers Party, the FREEZE backup, the three servers (dds, shaman and alibaba), each on its own magnetic tape storage device was donated [De Haan, 2014a].

Extraction of the data from the tapes turned out to be more challenging than expected. Finding the right hardware that was still operational proved challenging, especially an operational reader for the tapes. In July 2013, Henk Peek of the computer museum at the University of Amsterdam managed to successfully extract the data from the tapes [De Haan, 2016d].

In 2015, the course History of Digital Cultures (HDC) [of Amsterdam, 2014] was taught at the University of Amsterdam (UvA) targeting computer science students. During the course, a group of students opened up the contents of the FREEZE [Went, 2015]. Making the data accessible and doing some of the early explorations of the FREEZE, these students paved the way for students the following years to continue exploring DDS as part of the UvA course HDC.

In 2016, I followed the course History of Digital Cultures [of Amsterdam, 2015]. My team[1] explored the communication between users of DDS present in the FREEZE. By linking users via communication and removing weak connections using graph theory, we were able to discover communities inside the DDS using a data-driven approach [de Haan et al., 2017].

During our struggles with the data, we posed the question: "How hard could it be to get DDS working again?". A special project was started after the course called 3.0 operational [De Haan, 2016a], a play on the interface of DDS during the time of the FREEZE being version 3.0. The project was performed by a slightly different student team including myself[2]. In reconstructing the DDS from its original disks to run on new hardware, two versions of DDS based on the FREEZE were made in parallel: one team attempted to emulate the system to make the original code run. Actual emulation was eventually not achieved due to the Sun SPARC architecture not having good free emulators for the modern x86 architecture. Instead, an as close as possible approach was adopted to get the DDS working again, recompiling as much of the original source code as possible on x86 and reprogramming what did not work.

A second team built a replica, coding new software, to run with the original data. The results of the two approaches were surprisingly comparable [De Haan, 2016b].

The combined efforts in the RE:DDS project made it the winner of Digital Preservation Awards 2016, in the category The National Archives Award for Safeguarding the Digital Legacy [Coalition, 2016; van Gent, 2016].

[1]'Gabriele's Pizzeria': Kishan Nirghin, Tim Veenman, Millen Mortier, Robert Jansma, Randy de Vries, Ronald Bethlehem and Gabriele Di Bernardo.

[2]Marc Went, Robert Jansma, Ronald Bethlehem, Tim Veenman, Kishan Nirghin, Millen Mortier and Thomas Koch.

1.3 The digital archaeologist's tool belt

The archaeology performed on DDS required extensive knowledge of both the hardware and software involved. Knowledge of which tool to use and when to use it is key.

Many tools exist to analyse a variety of software [Lincke et al., 2008], but most of these require prior knowledge of the kind of software one is dealing with. A tool for analysing software metrics of C source code will have a tough time analysing a PHP website. In industry, there is a demand for such tools that work regardless of the input language [Rakic and Budimac, 2013].

With knowledge of the functioning and use of a system, finding the right tools, if existent, is possible. Alternatively, the system knowledge will allow the creation of the proper tools, since one knows where in the system to look for answers to relevant questions. But all of this assumes system knowledge.

What if one does not have extensive knowledge of the system?

By looking at the files in a file browser, such as the Windows File Explorer seen in figure 1.2, one may extract some basic information including the file names, extensions, size, date of creation, and some other metadata. In some cases a computer performing the investigation might recognise the type of file, allowing it to be opened with the appropriate type of software. Opening individual files is a manual process, a process infeasible when dealing with thousands of files. Tools that aggregate the (meta-) data and display them in an understandable format without requiring system knowledge are needed.

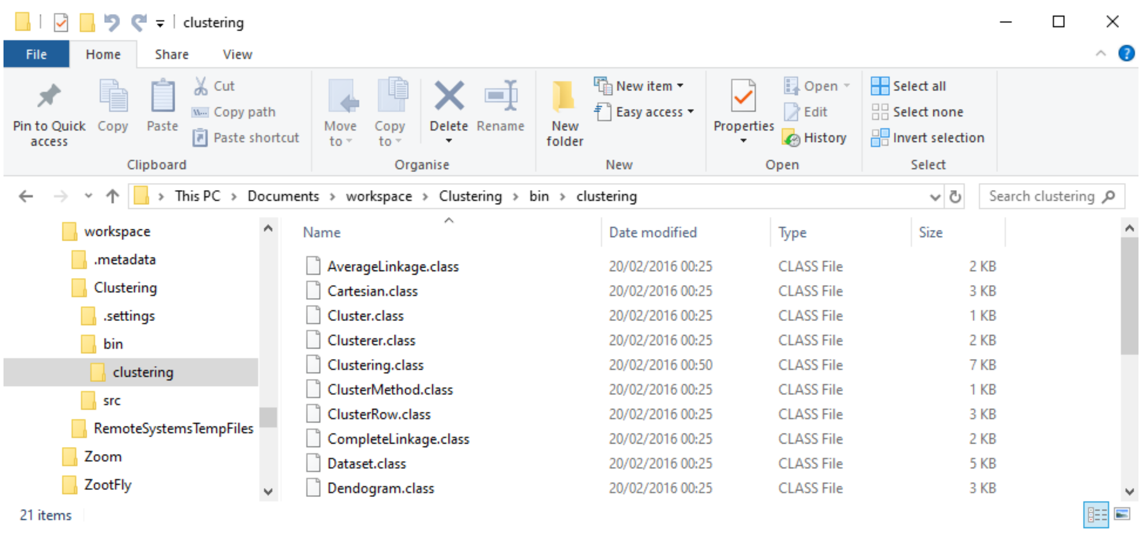
Some tools already exist like WinDirStat[1], "a disk usage statistics viewer and cleanup tool for various versions of Microsoft Windows". WinDirStat allows the user to get a visual representation of the files and folders and their size on a computer running a Windows operating system.

There are also programs like 'Count Lines of Code'[2] that can give insight into the number of lines of code on a computer system.

The problem is that there are hardly any tools to analyse digital artefacts that do not require prior knowledge of the system. This leaves researchers that do not possess extensive system knowledge without options to study their digital artefacts.

[1] https://windirstat.net/

[2] https://github.com/AlDanial/cloc



Name	Date modified	Type	Size
AverageLinkage.class	20/02/2016 00:25	CLASS File	2 KB
Cartesian.class	20/02/2016 00:25	CLASS File	3 KB
Cluster.class	20/02/2016 00:25	CLASS File	1 KB
Clusterer.class	20/02/2016 00:25	CLASS File	2 KB
Clustering.class	20/02/2016 00:50	CLASS File	7 KB
ClusterMethod.class	20/02/2016 00:25	CLASS File	1 KB
ClusterRow.class	20/02/2016 00:25	CLASS File	3 KB
CompleteLinkage.class	20/02/2016 00:25	CLASS File	2 KB
Dataset.class	20/02/2016 00:25	CLASS File	5 KB
Dendogram.class	20/02/2016 00:25	CLASS File	3 KB

Figure 1.2: Windows File Explorer

For archaeology, in general, the authenticity and integrity of the objects preserved are crucial. For digital archaeology, the authenticity and integrity issue is even more pressing. For digital archaeology, the questions of data integrity and authenticity may become conflated, even if they are different by principle. The integrity of a digital artefact concerns the preservation of the data that make up the artefact. Integrity can be achieved with bit preservation.

The question of authenticity is more complex. Authenticity deals with the functionality and experience of the artefact. Authenticity is very much dependent on the perspective taken when discussing authenticity.

1.4 Introducing metadata dating

The goal of the present research is to help researchers to understand what data they have at hand. There are hardly any methodologies and tools available that help researchers with such goals. That goal has been achieved by the creation of a methodology and tool applicable to any type of digital artefact without the need for any system knowledge, thus enabling archaeologists to ask and answer the right questions without possessing any specific system knowledge of their artefact system.

One data aggregate of particular interest comprises the time-related metadata. Using such metadata, dating can be performed [de Haan et al., 2017]. Metadata dating can be considered as "carbon dating" for files. Carbon dating is the method of dating organic objects in traditional archaeology by the proportion of C14 to C12 [Goh, 1991]. Temporal metadata serves a similar function in digital archaeology. The metadata of a file contain timestamps that show when the file was created and when it was last edited. For systems that use version control, it is easy to track development through commits, separate saves for each edit of a file. And even when no version control is used, it is still possible to say something about the growth of a system by inspecting when files were created. The last date files were edited on can be used to determine which files were the last worked on. Using metadata dating it is also possible to see over what period files were edited. By looking at the dating results and combining them with other secondary sources, events can be discovered or explained.

An additional step would be to retrieve all file extensions in the dataset. With some basic assumptions that hold for most cases, file extensions are a good identifier for the type of file. Using the extensions, files can be split up into categories such as pictures, programs, office files, source code and others. Useful data aggregates can be created per category. When dealing with images pixel counts can be relevant and for code the number of lines.

The variety of types of files is large. The task of correctly categorising every single file and recognising every type would create a problem of scale. However, it is fairly easy to prioritise some files over others using the number of times they appear in the system.

Since the present research focuses on the De Digitale Stad (DDS) dataset, a pragmatic approach has been taken as a starting point. Simply, the most common code file types present in the dataset have been chosen as a basis for categorisation.

There is much data available which may be harvested even without system knowledge. The present research aims to automate the process of collecting such data to facilitate research into digital heritage, specifically the time-related metadata. Ideally, digital resources become accessible to researchers without requiring extensive software skills and experience, and without presupposing the tacit knowledge required to recognise the types of datasets and the tools that go with the datasets.

1.5 Report structure

This report is structured as followed.

The field of software archaeology is discussed in Chapter 2. The chapter discusses what software archaeology is and discusses some tools that are available. The power of viewing information of data is covered and how data can be used to gain deeper historical insights. A model for viewing the preservation of digital artifacts is proposed. The model views preservation of digital artifacts from several different levels of abstraction and discussed how these levels might be used in preservation efforts. The research questions of the present research are discussed. The goal of this research is the extraction and aggregation of time-related metadata from digital artefacts and using these aggregations to gain historical insight into the artefact.

Metadata dating as a concept is covered in Chapter 3. Each of the time-related metadata and their historical value are covered. Distinction is made between metadata dating of individual files and systems as a whole. A more granular method of targeted metadata dating is discussed. The chapter culminates in the creation of an automatic tool, called MetadataDating, that can perform metadata dating and its design decisions are covered.

In Chapter 4, metadata dating is applied, with the use of the novel tool, to archives related to De Digital Stad. The provenance of the archives is discussed. The result of metadata dating and the insights gained into DDS by performing metadata dating into the archives are discussed.

Chapter 5 covers the conclusions of the present research. The value of metadata dating as a concept is discussed, the MetadataDating tool is placed in the field and considerations for how to apply it to artefacts are given, new insights in the theory of software archaeology are discussed and a future work section covers possible further avenues of study working upon the results of the present research.

2

The field of software archaeology

The primordial inspiration for the current research project is in webarchaeology, in particular in the recovery and study of the archives of DDS, the Amsterdam Digital City. Even so, the crucial aspects that make webarchaeology different, are issues with software. The tools which are so much in want, are the tools to deal with software. As a consequence, the generalisation is made from webarchaeology to software archaeology.

This chapter sets out to present a reconnaissance tour of the field of software archaeology. The concepts of software archaeology itself will be introduced. The power of computers as tools for historical research is considered, and how they can be used to gain deeper historical insights by placing their findings into historical context. The need for new, more comprehensive tools is explored and an overview of existing tools in the field is given. A model is proposed through which to view the preservation and authenticity of digital artefacts. Finally, the research question this research aims to answer are posed.

After so bravely generalising an advance note on software as text may be in place. Can't code, software, just be read? And can it not be interpreted as text? In principle, yes and no; in practice, no. The sheer size makes it infeasible to embark on a human "reading" of software. And more principally, software is meant to be "read" by other software and in the end by a machine. That is, it is technology, it should be functional. No reading of the code will allow one to truly experience the software [Alberts et al., 2017].

Isn't software supposed to be adorned with comments and documentation, specifically meant for human reading? Certainly. However, documentation for software is almost always outdated and inconsistent [Forward and Lethbridge, 2002]. That is, assuming it even is there, to begin with. Documentation that is available sometimes is a fully documented manual, other times a blog post and sometimes a comment hidden in the source code.

More importantly, though, precisely because it is technology the knowledge of what the software is and does is only earned through experience with a system. Such experience-based knowledge is gathered and exchanged in communities surrounding the software or system in question. Typically forums and blogs serve to ask for and share the intricate knowledge. And even then, not all knowledge on a practice of working with a piece of technology like a software system is made explicit and remains tacit knowledge. Working with complex software systems only deepens the difficulties of "knowing" software.

The above-mentioned difficulties are even true for the present report. Some of the claims are partially based on experience with computer systems and their inner workings. Good sources for the workings of software are hard to find because they are likely outdated by the time they are printed. Where possible, standard specifications are cited and otherwise other online sources are relied upon.

2.1 What is software archaeology?

The recovery, study and preservation of our cultural heritage has long been a field of research. There are all sorts of artefacts to be preserved and studied, each with its own set of characteristics, opportunities, and challenges. Works of art such as paintings require careful protection from touch and light, and occasional restorative efforts. Sometimes the medium carrying the artefact is replaced –with consequences.

Are oral traditions adequately preserved in writing and recording or will the transfer of medium not do justice to these traditions?

The effect of the recent fire damage to the Notre-Dame presents the restoration efforts with challenging dilemmas [Kincaid, 2020]. What damaged parts can be restored? What must be done with the remains of the parts that are beyond repair? How will these parts be replaced?

A new frontier in the study and preservation of cultural heritage is that of digital media. Computers and digital formats have been used in recent years to study and preserve non-digital artefacts [Daly and Evans, 2004]. Today, there is an ever-increasing need to preserve digital artefacts themselves [Brand, 1999].

There are three types of artefacts that are handled by digital archaeology.

Digitised artefacts are analogue materials that have been made digital. Examples are scans of letters, digital pictures of art installations and 3D models of buildings.

Born digital artefacts are created digital. They are originally intended to be accessed via a computer and often use the possibilities of the medium. Examples of born digital artefacts are websites, CD's and programs.

Reborn digital artefacts are born digital artefacts that have been collected and preserved in such a manner that they have been altered by the method of collection or preservation. Examples are compressed images, emulated video games and web archives [Brügger, 2016].

The term software archaeology was first claimed in the field of software maintenance [Cunningham et al., 2001; Hunt and Thomas, 2002].

> "[Software] [a]rch[a]eology is a useful metaphor: programmers try to understand what was in the minds of other developers using only the artifacts left behind. They are hampered because the artifacts were not created to communicate to the future, because only part of what was originally created has been preserved, and because relics from different eras are intermingled." [Robles et al., 2005]

The same metaphor is appropriate when trying to understand an artefact from a historical perspective instead of a maintenance perspective. Software archaeologists study digital artefacts, to understand how they work and to try to understand the mind of the developers of the artefact. Often only using the artefact itself as a source. These can be files, folders, entire systems and sometimes even hardware. Some software applications require specific pieces of hardware to be present to perform, a situation often encountered in video games. Video card manufacturers like to design software features that only work with their hardware. Examples include G-SYNC [NVIDIA, n.d.a] and Nvidia HairWorks [NVIDIA, n.d.b], with software features that work with specific Nvidia graphics cards.

It is commonly known that all software on computers is fundamentally made up of ones and zeros. But software is more than just the sum of its parts. Merely preserving the ones and zeros, called bit preservation, is not enough. Software is not mere data but also its functionality. A webpage can be stored as a screenshot, but a screenshot does not convey the functionality. Links no longer work, movies can no longer be played and the webpage now needs to be opened with a picture viewer instead of a web browser. Digital data are dynamic and should be treated accordingly. Not only should their bits be preserved, but also their functionality. It is a piece of technology [Alberts et al., 2017].

Besides the source bits and their functionality, files also come with metadata. These are data about a file that are not directly part of the file contents but contain information directly about them. The metadata of a file includes the name and location of the file, its size and the time-related metadata, among others. Metadata can provide additional context to a file.

Metadata can be accessed via a file viewer or the command line. On a Windows computer, one can open the File Explorer and see some of the metadata of files in the explorer itself such as when files were last modified. More metadata can be accessed by right-clicking on the file and clicking properties. The command line also offers utilities for accessing metadata of a file. Dependent on the type of metadata one wishes to access a different command is required.

2.1.1 Analysis and forensics

Imagine a scrapbook full of memories. Pictures of the past, old movie tickets and handwritten notes. The scrapbook can give one a lot of information about the person who created it, but one would have to manually look through the entire scrapbook to gain most of the insights.

In computer science, many things can be achieved by reducing everything back to ones and zeros. If one can store the scrapbook on a computer, one can work with it. Viewing all information contained in the scrapbook as data.

By seeing information as data and storing it as such on computers, one may make use of the tools available on computers. Data can be easily quantified, in bits, bytes, characters, counts or any other metric that can be assigned a numeric value.

A scrapbook can be taken and converted into data. The pictures become JPG files, the movie tickets can be put into a spreadsheet and the handwritten notes converted to text files.

Although this approach removes a lot of the charm and romance from the process, it allows one to use the computer to answer some questions far more easily than before. Counting all the pictures is as easy as retrieving the number op picture files. Finding out if and when a specific movie was seen is a simple search.

When looking at digital artefacts and analysing the data they contain, much can be learned from system administrators (sysadmins) and specialists in digital forensics. Many questions one can ask the data have been asked by sysadmins before. Because it is data, their queries and scripts can be reused to ask the same questions for different datasets.

Sysadmins can have tasks such as looking through log files and retracing the steps of a user on a system, in order to diagnose a problem or locate a fault. Analysing the activity and size of parts of a system are a routine part of a sysadmin's task, and these tasks can largely overlap with the interests of researchers of the system or similar systems.

In digital forensics, specific questions need to be answered by a system without compromising the integrity of the data and the metadata. Data integrity is of great importance in the forensics because the findings can be used as evidence. Any alteration would put into question the validity of the findings. A lot of parallels can be drawn between the field of software archaeology and digital forensics. Both require careful handling and preservation of digital material and research of the material.

The parallels and overlap between the tasks of a software archaeologist and the tasks of sysadmins and specialists in digital forensics speak to the interdisciplinary nature of the work of a software archaeologist. Software archaeologists require the data skills of sysadmins, the careful handling required in digital forensics while asking historical questions of the data.

2.1.2 From data to history

A data driven-approach will yield facts about one's digital artefact. The challenge is taking those facts and translating them into history. Is it even possible to gain insights from those facts?

To properly understand and appreciate an archaeological artefact of any kind, its context is needed.

An arrowhead, depending on its context, can have great historical value or very little. A modern mass-produced arrowhead is of no great historical or cultural value at present. An arrowhead from the middle ages would be far more interesting due to its age and relative rarity—even more so if the arrowhead can be linked to a famous battle or a specific historical figure. The historical and cultural significance depends not only on what can be learned from the arrowhead itself but also from where it was found and the written or oral history accompanying it.

The same goes for digital artefacts. Their stories and significance can only be understood through their context.

A question that could be asked of and answered using a digital artefact is: How many users did it have? A question that could be answered by the data, it is a data science question. A routine question asked of sysadmins. A sysadmin's tasks typically end when the questions go outside of the system itself.

How many users an artefact had, does not offer much insight into the implications of the answer. To gain those insights a different type of question needs to be asked. The number of users does not tell whether that is many or very few, relative to the standards of the time. The type of questions that deal with the implications of the answers offered by the data science questions is found in the media studies field. The number of users can be understood only by placing the number in the context of the system at that specific time. A website in the mid-nineties having tens of thousands of users would be an impressive number. Today, by contrast, a million visitors would not even come close to having the same significance.

The use of secondary sources is needed to fully understand and appreciate one's artefact. Digital artefacts come with their very own special sources. Digital artefacts have creators, often called developers. The creator can be a single person but also a group of multiple developers each with different skills and specialisations. Developers may not only tell how a system was created but also why certain design choices were made. Developers have almost a language of their own, shared jargon and tacit knowledge allowing them to effectively communicate about their craft with one another.

Digital artefacts also have users: people that interacted with the artefact. They provide a different perspective on the digital artefact. Unlike developers, users do not generally interact with the inner workings of a digital artefact, tending to interface only via the intended mechanisms created by the developers. A user of a website typically only interacts with that website through a web browser and has no direct interaction with the back end. Additionally, a community of hackers, modders, etc., often exists as well and their perspectives on artefacts can be quite different from those who use artefacts via intended means only.

These sources need to be placed in their context. How the person interacted with the artefact changes their perspective. The time that interaction took place also has its effects. Different coding languages and styles are common in different times. What was super fast in the '90s is now considered slow. These questions that take place outside of the artefact itself, provide a basis for a researcher to understand their artefact in context.

A software archaeologist crosses many fields, through the computer sciences and the humanities, always keeping a foot on both sides.

When handling data computer science skills need to be used while asking humanities questions. Once the data has answered what it can, a software archaeologist must rely on their humanities skills to place those findings into their historical context. When speaking to developers of an artefact, a software archaeologist can benefit from their computer science skills, allowing developers to speak about the artefact as if to a colleague.

Software archaeology is interdisciplinary work, requiring both computer science skills and humanities skills to adequately study digital artefacts.

2.1.3 Tools for software archaeology

In this section, the call for more tools in the field is shown and some tools in the field of software archaeology are discussed. This section aims to show the need for more tools and show how software archaeology is at the boundary between computer science and the humanities. The tools it requires are intrinsically boundary objects. The work to be done is truly interdisciplinary. A very succinct survey of the available tools is offered and discussed in the perspective of interdisciplinarity.

2.1.3.1 The need for tools

In the field of software archaeology, there is a great demand for tools and methods for performing software archaeology. In a discussion between leading experts in the field of Internet histories and computational methods, the need for more tools is actively discussed [Brügger et al., 2019]. Historians are becoming dependent on computer scientists to solve their problems involving computational methods [Crymble, 2015]. Failure to perform truly interdisciplinary work for both historians and computer scientists would result in failing projects.

Researchers in the humanities will have to become somewhat computer savvy from their side. The need for humanity researchers to gain computer skills is seen, but the task often seems daunting due to the high complexity of the tools involved and the often required tacit knowledge to even get started with some of these tools.

In their turn, computer scientists should offer their skills in a manner fitting the requirements of historical or archival research [Nielsen, 2019]. Part of making these skills attainable is lowering the barrier of entry by providing tools that do not require extensive computer and data science skills.

Highly modular, small, general-purpose tools with many possible configurations provide computer scientists with the possibility of stringing these tools together and altering them to fit their needs at the time. Flexibility comes at a price: complexity. Some of the complexity can be alleviated by creating tools that already have done the work of combining these smaller tools to fit the needs of humanity researchers, leaving only the configuration options directly relevant for their tasks.

2.1.3.2 Bitcurator

Bitcurator[1] is a collection of open source tools intended for collecting institutions, such as archives, museums and galleries. The goal of Bitcurator is to address two needs of these institutes: the ingest of digital material and the provision of access to the contents of said material [Lee et al., 2012]. Bitcurator would be used when an institution is presented with a new digital artefact for their collection or to further inspect an artefact in their collection.

Bitcurator has built-in tools for imaging and recovery of data from physical storage sources. One tool, Bulk Extractor makes reports of forensically interesting data, mostly by searching through files looking for email accounts, credit card numbers, phone numbers and other computer-recognizable pieces of data. A full list of the tools can be found on their Confluence page[2].

Bitcurator makes its tools more accessible by providing an entire environment in which they already work. In general, most software packages need to be downloaded and installed on an already existing system, which can require the user to solve any missing dependencies on that system. Bitcurator, however, comes with an entire virtual system. The Bitcurator Environment is a Ubuntu derived Linux distribution. It can be run as a virtual machine or installed as a live operating system. The individual tools could still be downloaded for those that want the freedom of managing their environment themselves, but the option of a fully working environment can make it much easier to get started using the tools Bitcurator provides.

[1] `https://bitcurator.net/`

[2] `https://confluence.educopia.org/display/BC/Tools`

Most of the tools themselves come in two versions. One version that has a graphical user interface (GUI), providing easy use and configuration of the tool. The other version is accessible via the command line, often with slightly more options than their GUI counterpart. The GUI approach lowers the barrier for those that do not have the skills or confidence required to work with the more complex command line version of the tools while retaining the flexibility for those who want it.

2.1.3.3 Count Lines of Code (cloc)

'Cloc - Count lines of code'[1] is a command line based utility that can count the lines of code in a file, directory, git repository and even archives. It has support for many different types of programming languages. The output displays the number of files, blank lines, comments and lines of actual code.

'Count lines of code' is built to work on many systems with the only dependency being a Perl interpreter, and even that not being required for the Windows executable. To make it even more portable, there is even a Docker image so the entire tool can run in a virtualised environment with all dependencies taken care of, besides docker itself naturally.

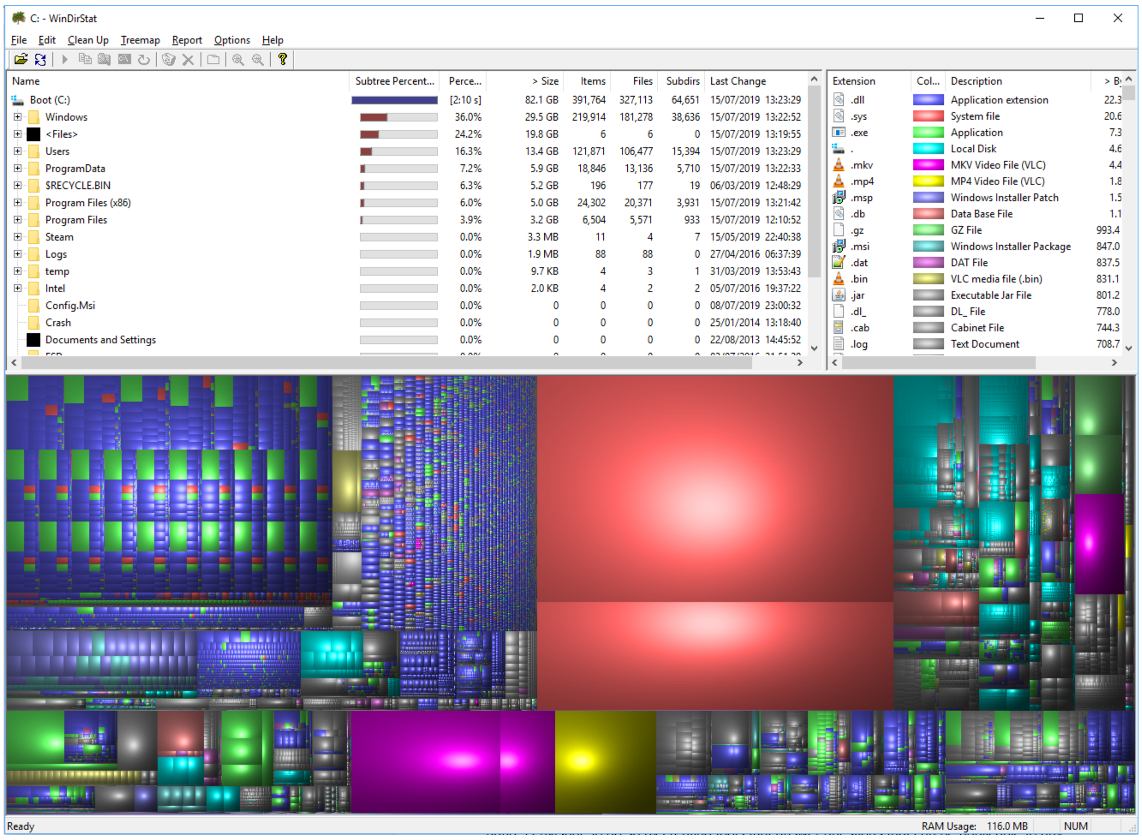
'Count lines of code' can run into problems with files that have overlapping extensions. It does attempt to compensate for the overlap for some known cases such as the .m extension, used by MATLAB, Mathematica, Mercury, MUMPS and Objective C. However, it is not always possible to establish what the programming language has been for cases in which there is not enough code in the files to make the distinction. Unfortunately, the tool does not have support for all languages. Some languages are simply still to new and obscure that they they have yet to been implemented, and some have other difficulties that prevent them to be supported [StephaneG31, 2019].

The overlap described above shows that a simple task such as counting the lines of code can already become very difficult without elementary system knowledge.

[1] `https://github.com/AlDanial/cloc`

2.1.3.4 WinDirStat

WinDirStat[1] describes itself as "a disk usage statistics viewer and cleanup tool for various versions of Microsoft Windows" , WinDirStat allows the user to get a visual representation of the files and folders and their size on a windows computer, see figure 2.1.



Name	Subtree Percent...	Perce...	> Size	Items	Files	Subdirs	Last Change
Boot (C:)		[2:10 s]	82.1 GB	391,764	327,113	64,651	15/07/2019 13:23:29
Windows		36.0%	29.5 GB	219,914	181,278	38,636	15/07/2019 13:22:52
<Files>		24.2%	19.8 GB	6	6	0	15/07/2019 13:19:55
Users		16.3%	13.4 GB	121,871	106,477	15,394	15/07/2019 13:23:29
ProgramData		7.2%	5.9 GB	18,846	13,136	5,710	15/07/2019 13:22:33
$RECYCLE.BIN		6.3%	5.2 GB	196	177	19	06/03/2019 12:48:29
Program Files (x86)		6.0%	5.0 GB	24,302	20,371	3,931	15/07/2019 13:21:42
Program Files		3.9%	3.2 GB	6,504	5,571	933	15/07/2019 12:10:52
Steam		0.0%	3.3 MB	11	4	7	15/05/2019 22:40:38
Logs		0.0%	1.9 MB	88	88	0	27/04/2016 06:37:39
temp		0.0%	9.7 KB	4	3	1	31/03/2019 13:53:43
Intel		0.0%	2.0 KB	4	2	2	05/07/2016 19:37:22
Config.Msi		0.0%	0	0	0	0	08/07/2019 23:00:32
Crash		0.0%	0	0	0	0	25/01/2014 13:18:40
Documents and Settings		0.0%	0	0	0	0	22/08/2013 14:45:52

Extension	Col...	Description	> By
.dll	🟦	Application extension	22.3
.sys	🟥	System file	20.6
.exe	🟩	Application	7.3
.	🟦	Local Disk	4.6
.mkv	🟪	MKV Video File (VLC)	4.4
.mp4	🟨	MP4 Video File (VLC)	1.8
.msp	🟦	Windows Installer Patch	1.5
.db	🟥	Data Base File	1.1
.gz	🟩	GZ File	993.4
.msi	🟦	Windows Installer Package	847.0
.dat	🟪	DAT File	837.5
.bin	🟨	VLC media file (.bin)	831.1
.jar	▨	Executable Jar File	801.2
.dl_	▨	DL_ File	778.0
.cab	▨	Cabinet File	744.3
.log	▨	Text Document	708.7

Figure 2.1: WinDirStat disk visualization.

WinDirStat is a tool used to identify what is taking up space on a computer. It is commonly used for clean up purposes, to identify files and folders that take up large amounts of storage space so that they can be deleted.

It offers a graphical representation of files and folders and the space they take up on the drive. The GUI allows for an intuitive interpretation of the results and makes it easy to see what files make up the system and where they are located in the file structure. More so than the typical folder structure and size displayed numerically.

[1]`https://windirstat.net/`

Additionally, the tool does some extension counting with identification of the extensions.

While WinDirStat is exclusivly availible on the Windows family of operating systems, similar tools exist for Linux (QDirStat[1]) and Mac OS (Disk Inventory X[2] or GrandPerspective[3]). Note that although they are operating system specific tools, one does not need to use a tool specific to the operating system being studied. A hard drive with a Mac OS based system can be studied using WinDirStat on a Windows machine.

2.1.3.5 Carbon dating the web

'Carbon dating the web' [Atkins, 2017] or 'carbon date'[4] is a tool that estimates the age of web pages [SalahEldeen and Nelson, 2013]. 'Carbon dating the web' is more relevant in the field of web archaeology, the study of the history of the web. It is worth mentioning because its goal is similar to that of the MetadataDating tool: namely, the dating of digital resources. 'Carbon dating the web' does dating for webpages.

'Carbon dating the web' offers its functionality via the command line or a web interface[5], making it very user friendly, although users are encouraged to install the service on their own machine as it is a computationally intensive service. Installation is offered in the form of a Docker image making it very portable.

The age of a webpage is traditionally determined by manually looking at the resource itself. Articles often carry the date of publication. Social media integration shows timestamps that can be used to estimate the age of the article. The age can also be estimated by examining the resource more in-depth, looking at the content and linking it to clues of the time it was published.

The process of finding clues to the date of publication can often be relatively easy upon manually examining a single web page.

The above mentioned manual process does not scale for larger datasets. Manually inspecting a couple of pages can be done, but inspecting millions of pages would require tremendous amounts of time and manpower. That is why 'carbon dating the web' utilises external sources such as indexes and web archives to trace the earliest mention of a web page. It shows the dating information for every source and displays the oldest source as the estimated age.

[1]`https://github.com/shundhammer/qdirstat`
[2]`http://www.derlien.com/`
[3]`http://grandperspectiv.sourceforge.net/`
[4]`https://github.com/oduwsdl/CarbonDate`
[5]`http://carbondate.cs.odu.edu/`

2.1.3.6 Version control

Not so much a tool for performing archaeology itself, version control is something to look out for in digital artefacts. When present on a system, version control is the holy grail of time-related metadata.

Version control is used in software development to track changes made to the software being worked on. Each alteration to the software is saved as a separate step called a commit. Each commit, if the developers are using the system correctly, contains the change made, when it was made, who made it and a commit message with an explanation as to why the change was made. Commits allow changes to be easily tracked. If a change introduces a bug it can be rolled back to a previous commit that does not have that bug. Going back to previous versions via commits makes it easier to trace back what change introduced the bug [Blischak et al., 2016]. There are many different types of version control implementations, a few popular ones are Git[1], CVS[2] and SVN[3].

Version control is only as good as the person using it. Each change needs to be tracked manually. The more granular the versioning process is, the more valuable the version control is. However, commits often include several changes across multiple files. Commits containing multiple changes make it harder to trace each change to separate times and increases the difficulty of retracing any bugs introduced to the software. In addition, the precision of the reason given in the commit message for the change is up to the discretion of the person making the commit. They can be as precise or imprecise as they want, or may even write irrelevant text[4]. Not having an enforced commit format can lead to overly complex explanations which impair comprehension as well as too simple explanations, which do not accurately describe what was changed. An example of such a commit message is "bug fixes", which does not go into detail what the changes were and why they solved any bugs but is sadly all too common.

The way commit messages are written is usually part of a community culture. Although valuable for yourself when reviewing one's own code, they are far more valuable in collaborative coding endeavours, where not only the one who has written the code needs to understand it but also the people that use or need to alter that code. In collaborative coding projects, rules are enforced on the quality of the commit messages. In projects

[1]https://git-scm.com/
[2]http://savannah.nongnu.org/projects/cvs
[3]https://subversion.apache.org/
[4]https://xkcd.com/1296/

using a ticket system, it is usually enforced to have a ticket number as part of the commit. Besides the valuable timestamps the commits provide, the commit messages can tell a lot about the internal history of the developers.

There are many tools available for tracking software development projects through version control. By tracking the commits it is possible to track problems being solved and features being implemented. These tools are mostly used to track progress during development and manage the project. They can, however, also be used to examine the development of a project after development has ceased. The insights used by project management during development to readjust to ongoing concerns can be used by historians to gain insight about the development. In addition, having access to all commits when development has ceased can give insight into what readjustments have been made and their effect.

2.1.3.7 Lessons from the tools

Tools valuable for studying digital artefacts are at the boundary between computer science and the humanities. That they are boundary objects is reflected in the tools themselves.

High portability is seen in Bitcurator, ‘Count lines of code’ and ‘Carbon dating the web’. These tools were clearly made to be used by different people on different systems. These tools remove the need for users to have extensive knowledge of configuring their systems to make the tools usable, by offering virtualised instances of their tools.

Bitcurator, ‘Carbon dating the web’ and WinDirStat come with GUIs. Having a graphical interface lowers the barrier for using these tools. Allowing humanity researchers to do the work without extensive computer skills.

Bitcurator, ‘Count lines of code’ and ‘Carbon dating the web’ offer command line access to their tools. Command line access allows for greater flexibility for those with the technical prowess to utilize the power of the command line. Having command line access allows computer scientists to automate the use of these tools for large datasets.

Specifically in Bitcurator and ‘Carbon dating the web’ one can see that they are the product of interdisciplinary work. They stand with one foot in the humanities and with the other in computer science. Not only because they are tools that are helpful when studying digital artefacts, but due to their clear computer science oriented interface with the addition of a more user friendly graphical interface.

In Table 2.1 an overview is given of the discusses tools and their features. The table shows if the tools have a command line interface (CLI), a graphical user interface (GUI) and offer virtualised versions of their functionalities.

Tool	CLI	GUI	Virtualised
Bitcurator	Mostly	Partially	Yes
'Count lines of code'	Yes	No	Yes
WinDirStat	No	Yes	No
'Carbon dating the web'	Yes	Yes	Yes

Table 2.1: Overview of tools and their features

In software archaeology the need for tools is clear. There is a lack of tools that fit the needs of researchers in this field. The interdisciplinary nature of software archaeology makes it that tools for this field need a broader scope than computer science tools, that are traditionally very narrow in scope. Additionally, the tools need to be operable without extensive computer and data science skills. The present research aims to provide a tool that fits the needs of software archaeologist.

A tool is created to extracted the time-related metadata that are covered in Chapter 3. The design of the tool considers the interdisciplinary nature of software archaeology, consideration in the design of the tool is discussed in Chapter 4.

2.2 Authenticity: Levels of preservation

Preservation of digital artefacts can be done on various levels. In this section, a model will be proposed explaining how one can think about the authenticity of digital artefacts by how they are preserved. In light of preservation, a digital artefact can be viewed and studied on different levels of abstraction.

Each proposed level is illustrated with an example from earlier work done with the DDS dataset demonstrating the value of preserving artefacts on that level.

2.2.1 The file level

The first level is the file level. In this scenario, the contents of the file or files are preserved. The files may be transferred to a different storage medium: as long as the files are intact and produce the same checksums[1], then they are considered preserved. The file level preservation is useful when only the contents of the file matter. The assumption is made that when the file will be accessed for use, the relevant software to read or otherwise work with the file is available and that the user has all the necessary context to understand the file. Assuming that only the content matters implies that the archivist and the historian know what they are doing in terms of software and context, and bear the responsibility for taking the artefacts out of context and regardless of the software running it.

> The file level strategy is useful for reinterpretations of the artefact. Using the file level strategy, fellow students and I parsed emails from the DDS reinterpreted the findings to produce graphs of email interactions. Using those graphs we were capable of discovering and showing communities being formed around a similar interest [de Haan et al., 2017].

2.2.2 The metadata level

The metadata level preserves not the files but also the metadata of the files. The metadata level includes the time-related metadata, filenames and other metadata. Metadata level preservation allows more of the context of the file to be preserved. Relevant software to read or otherwise work with the file is still required.

> The preservation of filenames including file extensions for the DDS dataset allowed for the automatic searching for specific file types. File extensions allowed for searches for specific functionalities such as webpages, which were vital for the reconstruction efforts [de Haan et al., 2017].

2.2.3 The file system level

The file system level stores the files, their metadata and also how the files are stored. The file system level includes the relative locations of the files. They must be stored in the same folder structure. Additionally, the file system in which the files are stored must be the same. A file system is how data is stored on a hard drive or another storage medium. FAT32, NTFS and APFS are all examples of file systems. File system level preservation allows for

[1]A checksum is the result of a calculation done on a file allowing for quick checking of the integrity of the contents of that file.

even more context than the metadata level. Relevant software to read or otherwise work with the file is still required.

> That the FREEZE was stored in the same structure as the running system allowed much of the functionality to be restored. Many pieces of code and HTML files rely on their functionality on their relative position to other files. Links in the replica of DDS worked because the relative positions of the restored pages were the same as on the original servers. File system level preservation made it so that all links that worked with relative position immediately worked on the replica [De Haan, 2016b].

2.2.4 The functional level

The functional level is where storage not only preserves the data but also the functionality of the software. Preserving a digital artefact at the functional level assumes that the software can fulfil its function, that the code can be made to work. On the functional level, the digital artefact should retain all its original functionality. Retaining functionality can be achieved either straightforwardly, saving hardware, data and software all in its original configuration, or absent of the original configuration, by making the code run on a new system: By migration to a different system, emulation of the original system or a different reinterpretation of the artefact. In functional level preservation, all digital context of the artefact is preserved. The relevant software that allows the artefact to function is included in the preservation and should work out of the box.

> In reconstructing the DDS from its original disks to new hardware, two strategies on the functional level were pursued in parallel: one team attempted to emulate the system to make the original code run; a second team built a replica, coding new software, to run with the original data. Both approaches preserved the functionality of DDS, only differing in how the functionality is achieved [De Haan, 2016b].

2.2.5 The hardware level

The hardware level is the most comprehensive level of preserving a digital artefact. Not only is the data, the code and the functionality preserved, but also the hardware. Hardware level preservation would imply the preservation of the entire original system. Of course, to keep an entire vintage system up and running is very difficult to sustain, as hardware will eventually fail, bitrot will occur and finding or creating the same hardware can become very costly and even infeasible as systems age.

Moreover, the system could eventually run in isolation only as other systems have their software updated all the time. In the case of preserving internet server systems, the exchange which was so characteristic of the internet is simply not achievable. In particular, one should not dream of having vintage systems interact with the present-day web. Besides compatibility problems with modern web browsers, an internet server that is not updated could be easily compromised by hackers using long known security exploits.

> Even on the hardware level, there is an example from DDS. Even though it is not a long term solution, entire servers have been donated and stored at the collection centre of the Amsterdam museum. These servers have already proven their value by providing the hardware required to make system images of all the donated servers, some of which were no longer operational when they were donated. Ensuring at least preservation on the filesystem level [De Haan, 2016c].

2.2.6 Applying the model

The above distinction of technical levels of preservation: File, metadata, file system, functional and hardware imply that the later levels preserve all the characteristics of the earlier levels. Overlapping of the preservation levels is not necessarily required when preserving artefacts from different level perspectives. One could, for example, create a replica of a system that has all the same functionality but works differently on a file level. An example could be the parallel approaches in the reconstruction of DDS. In fact, emulating systems of software by running them on a different platform is standard practice in software engineering [Tucker, 1965]. The difference between two functional level preservation efforts need not be discernible from a user's perspective.

Depending on the goal of the preservation effort, a view of preservation starting at the hardware or functional level could be desired. If the goal is to gain insight into how users interacted with the system, an approach starting from a hardware or functional level can achieve the goal more directly and possibly more effectively. Forgoing authenticity in different levels but not at the desired level, in the interest of cost, both time and monetary.

The proposed model provides a framework to view and plan the preservation of digital artefacts. The model can help researchers and collecting institutions identify their authenticity goals or where their preservation effort fall short. In the present research, the model is used to evaluate the integrity of the time-related metadata of the DDS dataset.

2.3 Research questions

The goal of the present research is to extract information from born digital artefacts, without requiring specific system knowledge of the artefact itself. By removing the need for prerequisite system knowledge of digital artefacts, the barrier for humanities researchers is lowered to perform the interdisciplinary work of a software archaeologist.

The hypothesis of the present research is that the extraction of time-related metadata, metadata dating, can be performed automatically on a digital artefact. The time-related metadata can be aggregated and visualised and new insights can be gained from these representations of the time-related metadata. All without requiring specific system knowledge of the artefact itself. In order to achieve this the question is split up in the several sub-questions.

2.3.1 Extraction

RQ1. Can the time-related metadata of files and folders be extracted?

In order to proceed, any time-related metadata present has to be extracted for each file. These are the *birth time*, *access time*, *modify time*, and *change time*.

For the extraction of the time-related metadata a tool was created called MetadataDating, referred to as 'the MetadataDating tool' to clearly distinguish between the tool and the methodology.

2.3.2 Aggregation

RQ2. Once extracted, can the time-related metadata be aggregated in a meaningful way?

Having all the time-related metadata of the files does not necessarily make it understandable. Each of the time-related metadata needs its own aggregate: for example, files per year, month or day.

These aggregations can then hopefully be displayed in tables or visualised as graphs.

RQ2.1. Can the aggregations of the time-related metadata be used to check if the metadata of the files are intact, ensuring a deeper level of data integrity than checksums do?

This question builds on and helps answer **RQ2**. As demonstrated in Section 2.2, the preservation of the metadata adds an additional level of preservation to digital artefacts. It is therefore important to ensure that the preservation operation is performed with the careful inclusion of metadata. Checking the preservation of time-related metadata is currently a manual process. Metadata dating can make it easier to confirm the preservation by aggregating the metadata in tables or graphs, allowing for easy search of unexpected dates.

2.3.3 Historical insights

RQ3. Once the time-related metadata are aggregated and made accessible, how can the aggregations give insight into digital artefacts?

The aggregations will present the time-related metadata as files per time period in tables and graphs. These graphs can have peaks and valleys that can be used to gain insights about the artefact.

If the aggregation of the time-related metadata be used to check if the metadata of the files are intact, then how does that integrity relate to the authenticity of the files?

Can one retrace periods of high or low production? Can one differentiate between the types of production taking place? For example, was code written in a specific month, or were office files created during that period?

These questions, which deal with the chronological order of events, are fundamentally historical by nature. They give an order to past events. Having an accurate order of events can open up the way to ask more historical questions. Once it is determined what happened, one can ask why it happened. If production increased or decreased, what was the cause of that? Why did certain file types get created and others not?

These questions go beyond the mere order of things and ask for the reasons behind them. To answer these questions, one needs to not only look at the metadata but also to consider the files themselves and include secondary sources to give context to those findings. One needs to perform the interdisciplinary work of a software archaeologist.

3

Towards tools based on metadata dating

This chapter presents the theory behind metadata dating, the dating of files based on their time-related metadata. New archaeological uses for the time-related metadata are introduced. Metadata dating can be performed on different levels. In the first section, dating is discussed on the individual file level. The second section considers how to use all data collected from the individual files and apply them to the system as a whole. It is explained how to perform more targeted metadata dating, such as targeting specific file types. Finally, a tool is presented that automates the process of performing metadata dating on software systems.

3.1 Metadata dating files

Computers are information processing machines. To do its job, a computer needs a way to store the information it processes. Computers store information by putting it in a file. We have all worked with many different types of files: text files, pictures, spreadsheets, and so on.

Consider a text file. The file is the text itself and nothing else. For a picture, the file is the values of all pixels in that picture. The information itself is the file. A file does not have contents, a file "is" the content. Two files with the same contents are identical files and will produce the same checksums.

The computer, however, stores more than just the information contained in the file: it also stores metadata.

Metadata is information about the file. Metadata includes the file name, its storage location, size and much more. These types of metadata are always stored for every type of file. However, some metadata is only present on some types of files. These are metadata that only makes sense for that type of file. For example, with pictures it makes sense to store the number of pixels and for music the length of the recorded sound. But for music, it does not make sense to store the number of pixels. These metadata can serve in two ways.

Firstly, metadata can make it easier to access information present in the file; for example, having a metadata field that stores the length of a piece of music makes that one does not have to read and process the entire music file each time one wants to check its length.

Secondly, holding information that is not present in the file may give one information beyond what can be learned from the file itself. For example, picture files often include a date on which the picture was taken.

One type of metadata that fulfils the second purpose and is the main interest of the present research are the time-related metadata. These metadata allows software archaeologist to perform metadata dating.

In classical archaeology, researchers can be working at a dig site of an ancient battlefield. They find human remains dressed in metal armour alongside a sword and wooden shield. One of the questions they would want to answer is: When did the battle take place?

To answer when the battle took place, the researchers can use a technique called carbon dating. Living things are made out of carbon, which has a particular isotope called Carbon-14. Carbon-14 is radioactive and decays at a (known) fairly slow rate. By taking the remains and measuring the amount of Carbon-14 atoms present in it, researchers can determine the age of the remains. The carbon dating results tell them when the battle took place [Goh, 1991].

Metadata dating does the same for files. By looking at the metadata of a file, the age of the file can be determined. The metadata may even contain timestamps that show when the file was created, accessed, when it was last edited, or when its metadata was edited. Even if a system does not have version control, these metadata can be used to say something about the age and evolution of a software system.

3.1.1 The time-related metadata

There are four types of time-related metadata all types of files may have: *birth time*, *ATime*, *MTime* and *CTime* [unixtutorial.org, 2008]. These times are expressed in seconds since an epoch [Group, 2018c], a set time in the past from which the counting started. In Unix-based systems, the epoch is traditionally 1970-01-01 00:00:00 [Matthew and Stones, 2008]. One thing to note about time-related metadata is that it is dependent on the correct functioning of the internal clock of the computer that registered the timestamp.

In the following sections, each of these metadata types will be discussed along with the information that can be learned from them on an individual file level.

3.1.1.1 Birth time

The *birth time*, sometimes called *crtime* or *creation time*, of a file, is the moment in time a file was created and is stored in seconds elapsed since the epoch. It can be used to calculate the time and date when an individual file was created. By using the seconds elapsed since the epoch, a known time and date, a new time and date is calculated. The *birth time* can be useful for several historical purposes.

The *birth time* can be used to gain insight into the timeline in which a software project was developed. Files with an earlier *birth time* will have been created earlier and so will probably have played a role earlier on in the development process.

Obviously, files would not have played a role before their *birth time* as they did not exist yet.

With a list of all *birth times*, some insights can be gained in the speed and scale of the development of a piece of software.

The *birth times* show when and how fast files were created and reveals during which periods relatively many files were created. Many files created in a time period could imply more development taking place. The assumption being that during development the creation of new files is a common task.

Likewise, the periods with relatively few files created could indicate slower development.

However, that assumption may not be true. It could also mean that the development was restricted to mostly already existing files. Other metadata, *MTime*, is required to test if already existing files were possibly modified during the researched period. Furthermore, without version control, it is impossible to determine what development exactly took place between the two timestamps. If the *MTime* is not during, but after the researched period, one cannot say if development took place during the researched period, only afterwards.

Sadly *birth time* is not always present in the metadata of files. Its presence or absence is determined by the filesystem. MacOS uses the Apple File System (APFS) as of 2017 and before that HSF+, both supporting *birth time.* Ubuntu (a popular Unix-like operating system) by default uses ext4 (fourth extended filesystem), while ext4 does offer a field for *birth time*, it is not fully supported by the core functionalities [Ubuntu, 2017]. There are ways to hack around the limitation [Ubuntu, 2014]. Providing one with access to the *birth time* but these methods are bound to change with future updates.

Note that the presence of *birth time* is determined by the filesystem. The same operating system can use different filesystems. The operating system does determine if and how users interact with the *birth time* if present.

3.1.1.2 ATime

The *ATime* of a file stands for its *access time* and indicates the last time at which the file was accessed. The access can be a user opening the file, making an alteration to it, using it via a different program or any other use of the file. The access does not necessarily have to be a user directly performing the access. A different program can access the files for its own purposes and update the *ATime* of that file. Accessing the file can be as simple as reading its contents, no modifications to the file itself are required.

ATime tells one when files were last accessed. The *ATime* can be used to determine what files were the last to be accessed in the system. The *ATime* can give clues to what the last actions were that have taken place on a system. The files with the most recent *ATime* will be the last files to be used. By sorting them one can retrace the final steps taken on the system.

There are also implications about the relevance of files when looking at the *ATime.*

Files that have not been accessed in a long time might have lost relevance to the use of a system since neither a user nor a different program has tried to access the file. The *ATime* tells one the moment the last interaction with the file took place.

Files that have been accessed recently would have greater relevance to the latest use of the system. A user or program actively accessed the file, so it must have had some use at that moment to explain its access.

3.1.1.3 MTime

The *MTime* of a file stands for its *modify time*, the time a file was last modified. Meaning that the actual content of the file itself has been changed or modified. For a text file, a change in *MTime* would mean that the text was altered. For a picture a change in *MTime* would mean that the picture was edited, for example, adding a filter or cropping the picture.

The modification does not necessarily have to be a user directly performing the modification. A different program can modify the files for its own purposes and update the *modify time* of that file.

Changes in metadata do not result in a different *MTime*. *MTime* remains unaltered because it is not a modification to the file, so the file would not be modified and therefore keep its *MTime*.

MTime tells one when files were changed for the last time. If a file is changed, that would imply active development of the file. Code would be altered, pictures changed, text edited.

Using the *MTime* it can be determined when active changes were made to files.

Recent changes would imply active development. The files with the most recent *MTimes* would be the last files that would have been worked on. *MTime* can tell one what the last part of the system would be that received changes.

Inversely, no recent changes taking place would imply that development for that file had ceased. When a file has not had its *MTime* updated in a long time, then no changes have been made since that time.

If the *MTime* and *ATime* of a file are equal, then one knows that the last time the file was accessed its contents were changed. Modifying a file requires it being accessed, so changes in a file, results in both the *MTime* and *ATime* being set to the current system time.

3.1.1.4 CTime

The *CTime* of a file stands for its *change time*, the last time the file or its metadata have been changed. It is different from *MTime* in that it does not only concern the content of the file but also the metadata. Every time the *MTime* is updated because of a modification to the content of the file, *CTime* is also updated to that same time. When a piece of metadata is altered only the *CTime* is updated, not the *MTime*. For example, giving a user permission to access a file. Permission information is stored in the metadata of the file. By updating permissions the file's contents remain unchanged but the metadata would be altered. Altering permissions would result in only the *CTime* being updated.

CTime is a close relation to the *MTime* and the insights gained from it are dependent on the *MTime* of the file.

If the *CTime* is equal to the *MTime* then the last modification to the file has been its content, and the metadata would have been updated with the modification of the file. If the *CTime* is equal to the *MTime*, the insights are restricted to those of the *MTime*.

If the *CTime* is different from the *MTime*, more insight can be gained. A different *CTime* and *MTime* means that the metadata of the file has been altered but not the content of the file. The alterations could be changes to the permissions of a file, a name change, or any other alteration to the metadata. Although one does not know what the alteration was, there is a strong likelihood that the alteration is related to some system administration task. Depending on the type of file and its location on the system, estimated guesses could be made, given enough context and system knowledge.

3.1.1.5 Exceptions

Above it has been discussed how the time-related metadata changes as files and their metadata are accessed and altered. At least, the discussion so far has covered how they are intended to function. As is often the case with computer software, there are some exceptions and quirks that result in different results than expected.

This section discusses some of these exceptions. Note that the examples in this section are not an exhaustive list, but they illustrate why the metadata do not always act as expected.

Touch

In some cases, system administrators will "`touch`" a file using the console command `touch` [GNU, n.d.c]. `Touch` updates the time-related metadata of the touched file [GNU, n.d.a]. They can set the values to specific timestamps or by default update all time-related metadata to the current time. Using `touch` would result in an *MTime* that does not correspond to the time the file was really updated for the last time.

System admins will use `touch` when they are testing or using a functionality that uses the time-related metadata. For example, they could have a script that periodically checks a folder for new or updated files and makes a backup of those files to a different computer. If the backup process goes wrong in some way, perhaps the Internet connection going down during the backup process, then the backup is faulty. They can invest time in making the script do extensive checks on the files in both the source and destination, ensuring data integrity. Making such a script would cost them some time to set up and test it. They could instead simply reset the metadata with a single command and let the original script make the backup again.

Command implementations

The way commands are implemented can have major implications for how the time-related metadata are affected.

Consider the example of moving a file. In a Unix-based terminal moving files is done with the `mv` [GNU, n.d.b] command. In Ubuntu, `mv` changes the *CTime* and leaves *ATime* and *MTime* untouched. The behaviour in Ubuntu is what one would expect as the file itself does not have to be accessed or modified. Only an alteration to its location metadata is needed. MacOS when using the same command has a different result. On macOS, all time-related metadata is reset to the current time. Resetting all time-related metadata implies that the file was not moved, but a new copy of the file was made and the old file was deleted.

The differences between Ubuntu and macOS is not merely a difference between operating systems but a difference in implementation that can also happen within operating systems. In macOS when a file is moved in the Finder instead of the Terminal the *ATime* is altered but the *MTime* and *CTime* are unchanged. The changes in *ATime* but not in *MTime* and *CTime* is completely unexpected behaviour and shows that the metadata's integrity is dependent on the implementation of the commands used.

Birth time and copies

One implementation that differs on systems is how *birth time* is handled when making a copy of a file. The difference is not due to unexpected behaviour caused by the implementation. It is due to a fundamental difference in looking at a copy of a file.

As a historian of digital artefacts, one would want a copy to be a perfect copy. A perfect copy being: With all of the metadata, including *birth time*, exactly the same as that of the original file. And some implementations of copying do copy over the same birth time. However, there is a different way of looking at a copy of a file. As a copy, thus not as the same file. Although the contents are the same, new memory has been allocated for the new copy and so it could be seen as a fundamentally new file with its own *birth time*. There are implementations of copying that do follow both approaches [apple.stackexchange.com, 2015].

ATime vs performance

In the previous sections, it has been demonstrated that the time-related metadata have value, not only for archaeological purposes but also for sysadmins and the eforensics. From an archaeological point of view one would like to have these data as complete as possible, in the real world, the cost can also play a role. Real-world applications have a decision to make as to the value of these metadata and their performance cost.

To update the time-related metadata it has to be written to the storage device. this is a relatively time intensive [Schmuck et al., 2002] and power [Garrett, 2008] costly operation.

In the case of *MTime* and *CTime*, normally these values are always updated, because of their potential uses and because they are not updated as frequently as *ATime*. *MTime* and *CTime* are only updated when actual changes are made to the data or it's metadata. Significant events where something about the system changes. There are cases to be made to also reduce the number of times they are updated [Son et al., 2017].

On the other hand *ATime* is traditionally updated on each access, regardless of changes being made to the data or metadata. In practice files are opened or otherwise accessed far more often then they are changed, resulting in more performance being spent on updating *ATime* than the other time-related metadata.

In practice, *ATime* is updated often updated lazely instead of each time a file is accessed [GNU, n.d.a]. Although relatively costly at roughly 10ms for a hard drive [Microsoft, 2014], in normal uses one would not notice the extra time spent. However, there are cases where each little bit of time can have an effect. For example software for high-frequency trading on the stock market. These pieces of software rely on being able to trade often and fast. Each fragment of a second later could mean that another trader could have already bought the stock it was intending to buy [Goldstein et al., 2014]. All the time that was spent updating metadata could have been spent on trading, resulting in more profit. In the case of trading, among others, where speed is important, the system administrators can choose to update the *ATime* only in specific conditions or even not at all, in order to speed up operation.

3.2 Metadata dating systems

Metadata dating results of individual files may be used to gain insights about the whole software system. This section covers how time-related metadata can be used to gain insight into digital artefacts as a whole, as opposed to the artefacts individual files.

3.2.1 Metadata as a check of integrity

The time-related metadata can be used as a check for the integrity of the file and its metadata.

When dealing with files for which it is known that they were created at or before a certain date and any of the time-related metadata are past that date, then one knows the metadata has been compromised. For example, if one has a system backup that was made in 1998 and there are *birth times* in the backup later than that date, then the integrity of the backup has been compromised. Changes have been made to the backup after it has been made. These changes can be the cause of incorrect copying of the backup, intentional or unintentional changes being made to the backup and even bitrot.

Although checksums of the images are the preferred method of testing file integrity, metadata dating does offer a quick check if the checksums are skipped or even before the checksums are made.

When a backup is made, *birth time* can be used to check if the metadata has made the transfer intact. Afterwards, the checksum can be made, being sure that the copy preserved the metadata.

3.2.2 Aggregating metadata

The true power of metadata dating comes from its ability to take a step back from individual files and gives one insight into folders and systems as a whole.

Having the time-related metadata for a file it is possible to say when the file was created, last accessed, last modified and had its metadata altered for the last time. These bits of information give one valuable insight into the files themselves.

Files do not exist in a vacuum and most of the time they are part of a greater system. Knowing that the files have some relation to each other, the time-related metadata can give greater insights into their use and how they relate to each other.

3.2.2.1 File sorting

The easiest way to aggregate the time-related metadata is to make a list of all the files. The list can then be sorted by the dates of their time-related metadata.

A list of all *birth times* would result in the order in which files were created. The order in which files are created allows one to retrace the development of a system. It shows one when files were created and what other files were created around that same period. The birth times give insights into what files might be closely related in functionality, since files created in the same time period will likely serve the same greater goal.

On starting a new project often a new folder is created and is quickly filled with all sorts of files for that project, all created relatively close to each other. When later that same project is expanded upon with new functionality, the new files required for that functionality are likely created more closely to each other than to the already existing files.

Alternatively, it can show different goals being pursued at the same time.

Files being created in the same period does not necessarily mean that they are related in functionality. Relationship in functionality can only be determined by closer examination. It does, however, tie the files together in time. Their creation was at the same time so they were likely worked on at the same time.

A list of all *ATimes* would result in the order in which files were accessed. The order of latest accesses of files allows one to retrace the most recent steps taken on a system.

If one sees a lot of images being accessed one can infer that the user might have been taking a look at them. If they all are part of the same folder perhaps they were going through an album. If the images are all in different places, perhaps the user was looking for a specific picture. Taking a look at the pictures might give more insight. Thanks to metadata dating one now knows where to look.

A caveat is in place, as the *ATime* is overwritten with every access. The change in *ATime* would make it impossible to determine in what exact order files were accessed. If a file was accessed multiple times only the latest access would still be recorded.

A list of all *MTimes* will give the order in which files were modified for the last time. The order of final modifications would allow one to retrace the latest modifications made on a system.

The last modified files have had some form of development on them. Taking a closer look at those files may reveal what was being worked on in the system. The location in the file structure can help determine if the modifications serve a similar purpose or are perhaps different projects being worked on simultaneously.

The same warning as in the case of *ATimes* must be repeated here, since the *MTime* is updated with every modification. The alteration to the *MTime* would make it impossible to determine in what exact order files were modified. If a file was modified multiple times only the latest access would still be recorded.

A list of all *CTimes* would result in the order in which files were changed. There would be an overlap with the list of the *MTimes*, these points will provide no further insight besides the insights already gained from the *MTimes*. The files that do not overlap do as they have had their metadata altered but not their contents.

If a group of files has had their *CTimes* altered simultaniously, resulting in equal *CTimes*, and the *CTime* of each file is not equal to its *MTimes*, than this group was most likely part of a sysadmin task. The types of files, their metadata and their location could give some clues to what the change might have been. For example, a website that has a bunch of HTML files with the same *CTimes* and the *MTimes* of those files are recently before the *CTimes*, this could indicate that the webpages were made available online after the modifications.

As with the other metadata, *CTime* is updated with every change so it does not record multiple changes. The volatile nature of time-related metadata has to be considered when trying to interpret the implications of the *CTime*.

3.2.2.2 Visualisation

Each of the sorted metadata lists could be represented visually as the number of files with time-related metadata in a time period. For example, the number of files in a year by *modify time* as can be seen in Figure 3.1.

These visualisations can show peaks and valleys in activity. A closer examination of the files in those peaks or valleys could give insight into the possible reasons why the highs and lows may be the case.

External sources might claim high or low activity on the system at times, these visualisations can be used to confirm those claims.

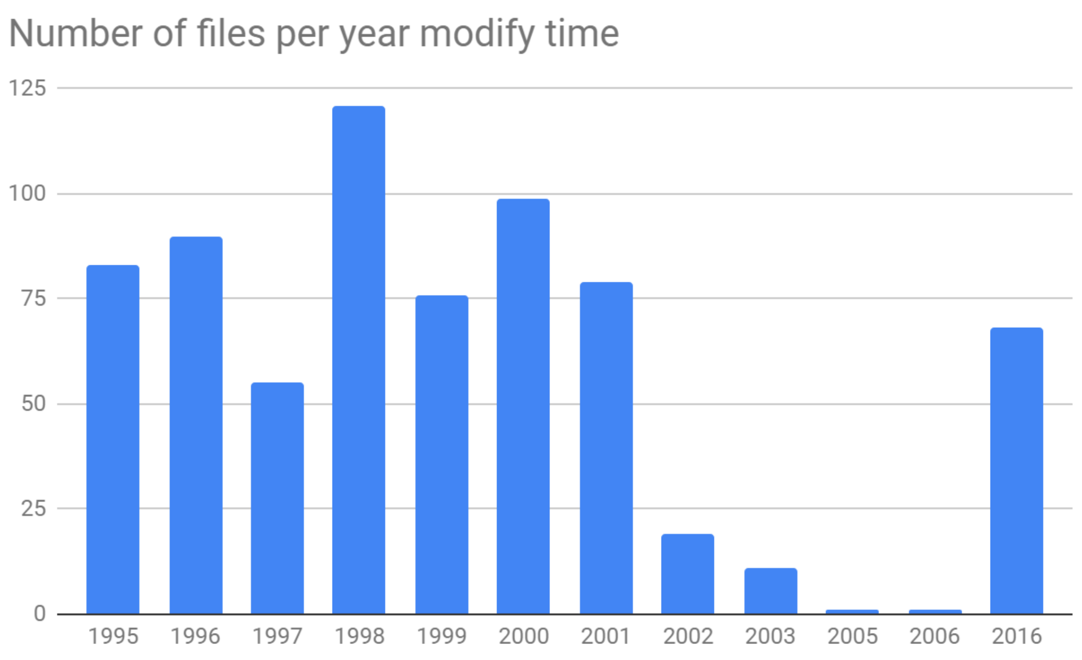


Figure 3.1: Number of files per year by *modify time* in the Kermit archive.

3.2.2.3 Interactions

Comparing the visualisations could offer additional insights.

A peak in *MTimes* that is far back enough can signify the completion of a large amount of related work. A final crunch to get a project finished. If the peak in *MTimes* is followed by a peak in *birth times*, one could hypothesise that after completing one project, a new one was started.

Comparing *birth time* and *ATime* can give some insight into how long files stayed in use. Similarly, *MTime* and *ATime* can give insights into the use of the files.
ATime will always be at the same time or later than *MTime*. But how much later can give an indication of how long it stayed relevant after its last modification. The visualisation can give an indication of the differences between *MTime* and *ATime* on a system level. Did files stay in use for long or short periods? The more surface area the *ATime* visualisation has at later dates compared to the *birth time* and *MTime*, the longer files stayed active on average.

The comparison of *MTime* and *CTime* visualisations can give insight into the number of metadata changes that took place on the system. If these visualisations are the same then *CTime* is equal to the *MTime*. When the visualisations differ then there are files that have had their metadata altered after their latest modification. The greater the difference in the visualisation the more files have had their metadata altered after their latest modification. Large differences between *MTime* and *CTime* could imply massive system administration tasks took place.

3.3 Targeted metadata dating

Until now metadata dating has been discussed when performed on individual files or all files in a system. Metadata dating allows one to get insights into individual files or the system as a whole.

A great way to gain additional insight into the activity on one's system is by increasing the granularity at which one looks at one's dataset. By having more specific subsets, more specific questions can be asked using metadata dating and more direct implications as to what may be the answers to those questions.

By performing metadata dating on specific folders one decreases interference from files that are less relevent to the research being performed. For example, on a Windows machine one could be interested in only the files belonging to a specific user. These are usually located in a folder with the user's name in the "Users" folder of the "C:" drive.

In most cases, the files related to Windows itself may not be of interest to the researcher. In that case, the "Windows" folder can be excluded from the dating request.

However, some system knowledge is needed to know what folders to include or exclude. Knowing what folders to use becomes even more difficult when the user of the system uses the folders in a different manner than standard. For example, by putting pictures in the "Documents" folder.

Another way to split up one's data is by file type. The great thing about looking at file types is that it is system-agnostic. All commonly used operating systems identify file types by a file extension. A file extension is the last part of a file name, that comes after the dot. A few examples are 'txt', 'doc', 'jpg' and 'c'. These are text files, Word documents, JPEG pictures and C source code respectively. The computer uses these extensions to determine what program to use to access the file.

The extensions can be used to determine the type of file. A list of file types can be created that the researcher is interested in. For example, a researcher interested in pictures can limit their search to only include files with the extensions jpg, jpeg, png, gif or bmp. Note that using extensions does require knowledge of the extensions commonly used by images and the above list is far from complete.

Still, using extensions allows one to create lists of common extensions that serve a grouped purpose. It does not require specific system knowledge to create such a list, only general knowledge of the extensions used by the types of files the researcher is interested in.

Some examples of the groupings that can be made using extensions are picture files, movie files, sound files, office files and code files.

These groupings can be made beforehand using common lists of extensions used for a purpose, or they can be made after inspection of the digital artefact. Getting a list of all extensions and how often they occur can be accomplished with a single line of Bash code or by counting occurrences in the spreadsheet made by the tool for automatic metadta extraction discussed in Chapter 4.

Knowing what extensions there are in a digital artefact, one can group these by whatever metric makes sense for the questions being asked of the digital artefact. For example, when looking at a website, it may be helpful to see the development of the website not only through the HTML, PHP and other file types that make up the frontend of that website but also include the backend code, logs and other files that do not directly interact with the end-users of the website. The more inclusive approach takes a look at the development of the website in a larger scope.

It may, however, make more sense to only zoom into the frontend files, giving a more narrow but focussed look at those files. What part is looked at depends on the question one is asking from the system. Questions relating to the end-user experience will have more value from only the frontend files, while questions about the system as a whole will prefer a more inclusive approach.

Depending on the digital artefact and the questions asked from it, different groupings can make sense. These groupings will be shared by similar artefacts or artefacts that have similar questions asked from them.

3.3.1 Difficulty using extensions

While extensions provide a good way of identifying file types, there are still some challenges using extensions as a basis for categorisation.

More common extensions are usually only used for a commonly known purpose. Files using the jpg extension will almost always be a picture file, because the jpg extension is so well known by the average user as a picture. To use the jpg extension for a different purpose would be confusing for most users because they are already familiar with the extension. Having some extensions being common knowledge protects the commonly used extensions from having overlap between different types of files.

Problems can arise when extensions are less common. They are more difficult to identify due to their uncommon purpose. Additionally, there is a greater risk of the extension having overlapping purposes. One example is a file using the m extension. The m extension is used by both MATLAB and Mathematica files. Digital artefacts which have the m extension would require more in-depth analysis of these files to identify which type of file it is. ‘Count lines of code’ has some built-in functionality to identify these files based on their content. Identifying files by content is, however, not always possible and becomes more difficult the smaller the files and the more file types use the same overlapping extension.

There are also files that do not have any extension. These files are impossible to identify using the extension. They often serve a limited bespoke purpose. To identify the purpose of files without extension, it is needed to manually analyse these files and the files that may access it.

3.4 The MetadataDating tool

A command line tool, MetadataDating, is created that automates the process of metadata dating. The tool is referred to as ‘the MetadataDating tool’ to clearly distinguise between the tool and the methodology. To create the MetadataDating tool several technical design choices needed to be made. In this section, those design choices are justified.

When developing a tool it is important to choose a coding language that can serve the end goal of the tool, in the case of this research, to extract and aggregate the time-related metadata of files. The choice of programming language can also influence how the end-user interacts with the tool. It is advisable to keep the form of interaction similar between tools. Similar interaction helps the end-user to get familiar with the type of tools and not having to relearn how each tool works. Because of these reasons, it was decided to make the MetadataDating tool in Bash [Foundation, 2017]. Bash is a command language interpreter and programming language used in many data analysis tasks when the dataset is computer systems themselves. System administrators use Bash for most of their tasks. It is exactly these types of tasks that have a large overlap with the tasks of a digital archaeologist, retracing the steps of previous administrators and users.

Another reason Bash was chosen is because Bash is a very powerful command line scripting language that allows the user to utilise any tools installed on the system that can be operated via the command line. Command line access makes it a very versatile tool, allowing other command line tools to be combined into a single script. To make sure that the MetadataDating tool is usable on other systems it must utilise tools that are supported on most Bash capable systems.

Other tools the digital archaeologists use are also quite often written in Bash or like Bash are accessed via the command line. Although there is a desire for more user friendly interfaces that utilise a GUI, when these do exist, they offer less control to the researcher than their command line counterparts. Therefore MetadataDating's development prioritised the development of command line functionality and does not currently offer a GUI.

Portable Operating-System Interface or POSIX [Group, 2018a] is the leading standard to describe how Unix-based Application programming interfaces (API's) should work, including the command line. It describes the fundamental functions and their behaviour of Unix-based systems. Most Unix-based operating systems follow its specifications. Following the POSIX standard allows programs written for one system to also work on any other system if the code is built using POSIX functions. POSIX is partially or completely supported by most Linux based operating systems, macOS and even has built-in support in Windows 10 with the Fall Creators Update, that introduced the Windows Subsystem for Linux [Raj, 2017].

In the POSIX standard, there is one command called `find` [Group, 2018b]. The `find` command describes how files can be found inside of POSIX compliant systems. Its simplest uses are to locate files with a specific name or in a specific location. `Find` is, however, a very versatile tool allowing for far more specific searches. One of the ways files can be filtered using `find` is by their time-related metadata. Specifically their *ATime*, *MTime* and *CTime*. These attributes can also be displayed by `find`. Due to the functionality of *find* to display the time-related metadata, `find` is the backbone of the MetadataDating tool.

With *ATime*, *MTime* and *CTime* extracted, only *birth time* needs to be extracted from the files. *Birth time* is not as commonly available as the other time-related metadata. *Birth time* not yet being part of the POSIX standard. There are implementations of `find` that do support *birth time*, but due to there not being a ruling standard for this functionality with `find`, using `find` is rejected. Different implementations can use different syntaxes to complete the same function. Different syntaxes would result in working code on one system and not working code on a different system, with the same code.

One option would be to set a specific version of `find` as the standard to be used for the MetadataDating tool. Although using a specific version of `find` would result in a clean implementation, enforcing a specific version is not wanted as `find` is a core functionality of any Unix-based system and it would be inadvisable to use a different implementation than belongs with the operating system. There may be version-specific implementations that are used by the operating system. By replacing `find`, these functionalities may break and create unexpected bugs elsewhere in the system.

Alternatively, although not part of the POSIX standard, there is a command called `stat` that does offer the functionality of displaying the *birth time* of a file. `stat` is a command line tool created to display the status of a file or file system. It can display basic facts about files, such as their name, location, size and much more such as the *birth time* and the other time-related metadata. `Stat` is a popular command available in many operating systems. There are, however, different specifications and implementations of the command. FreeBSD and GNU Coreutils[1] both have implementations of `stat` with the same main use but different implementations and syntax. Although different versions of the command exist, these versions are more likely to remain working on their respective systems than an implementation of `find`. Because `find` is part of the POSIX standard, it is likely that it will eventually include *birth time* in its specifications, causing all implementations to switch to the chosen syntax standard. If the tool uses a different syntax then the tool will cease to function on systems that use the new POSIX standard until updated. `Stat` is chosen to provide the *birth time* due to its wide availability and because it is not part of

[1]The GNU Coreutils stat command claims to have this functionality but is in fact blocked by kernel support of the xstat() interface [Blake, 2011; pixelbeat.org, 2012]

the POSIX standard, making it less likely that implementations will switch the syntax of the utility when updated.

The MetadataDating tool gives a TSV (tab separated value) file as output. The TSV format was chosen because it is essentially a spreadsheet and can be read by all spreadsheet software, such as Microsoft Excel, Google Spreadsheets or LibreOffice Calc. Researchers can use their own preferred tools for further analysis of the dating results.

The resulting file consists of 9 columns.

File: The full path to the file.

Access Time: The last time the file was accessed in a human readable format.

Modify Time: The last time contents of the file was modified in a human readable format.

Change Time: The last time the metadata of the file was changed in a human readable format.

Access Time(epoch): The last time the file was accessed in seconds since the epoch.

Modify Time(epoch): The last time contents of the file was modified in seconds since the epoch.

Change Time(epoch): The last time the metadata of the file was changed in seconds since the epoch.

Birth time: The time when the file was created.

Extension: The extension of the file.

Each line of the file shows the results for a single file or folder in the tested dataset.

The full MetadataDating tool and user documentation can be found on:
https://github.com/smootyboy/MetadataDating

4

Metadata dating DDS case study

In this chapter, a case study is presented, metadata dating via the newly created tool is applied to a dataset related to the 'De Digitale Stad' (DDS). DDS is a website of the early Dutch web. De Digitale Stad dataset is discussed, providing the context needed to understand the results and the higher-level interpretations of those results.

4.1 The Digital City dataset

It is important to discuss the origins of the dataset and how it has been stored and used over the years. The provenance of the dataset is needed to provide explanations for any possible unwanted alterations to the metadata of the dataset. The interactions with the individual archives can leave their mark on the dataset.

In 1994 DDS was launched. On its two year anniversary in 1996, the FREEZE was made. The FREEZE was a backup of the dds, shaman and alibaba servers that comprised DDS.

In 2001 DDS ended its free services.

In 2011 the RE:DDS project was launched beginning the archaeology of DDS. At the start of the project, the Grave Diggers Party was held, a four day event for the donation of material related to DDS. During the Grave Diggers Party, several digital sources were retrieved.

For the present research, two of those sources will be explored using metadata dating.

The first source is The John archive. The archive was provided by John Haley and contains backups of parts of DDS, including source code.

The second source is the Kermit archive. The Kermit archive was constructed by combining the donations of multiple people's personal archives. The Kermit archive includes many different DDS projects and logs.

Shortly after the Grave Diggers Party, the FREEZE was donated. Each server on its own magnetic tape storage device.

In 2013 the magnetic storage tapes of the FREEZE were successfully read.

The FREEZE archive existed out of a compressed archive file in the tar gzip (.tgz) format containing the data readout of the magnetic storage tapes.

In 2015 The FREEZE archive was used in the course History of Digital Cultures (HDC) taught at the University of Amsterdam in January. The original archive file was saved, while a copy was used to extract the data from. The extraction resulted in a massive explosion of the size of the archive. The result of 6 files in the archive being corrupted and filled with nonsense data. A copy of the archive was made with the contents of those 6 corrupted files removed to keep the archive manageable. The copy without the corrupted files, was placed on a server and used to perform research on by the students. Each following year until 2020 the extracted copy without corrupted files was used and maintained for the course.

For the present research one of the servers stored in the FREEZE, dds, will be explored using metadata dating. The server will be referred to as the dds server, without capitalisation, whereas De Digitale Stad itself will be referred to as DDS or De Digitale Stad, with capitalisation.

During the present research, the same copy as used during the HDC course was initially used. Due to the results obtained from the copy of the dds server used by HDC, metadata dating was also performed on the original extraction of the dds server with corrupted data. The additional dating of the dds server will be discussed in the results section.

The Kermit and John archives were not part of the course and were stored separately. In 2016 a project was done to consolidate all data on DDS into a single server. The John and Kermit archives have been copied over from their storage devices and moved to the server. The versions of the Kermit and John archives stored on the consolidated server were used in the present research.

Resulting in a total of three archive sources to be used as a dataset for testing and showing the effectiveness of the MetadataDating tool. The John archive, the Kermit archive and the dds server.

The results of the MetadataDating tool have then been aggregated using spreadsheet software, for this research Google spreadsheets. The time-related metadata have been counted for a manually determined time period. Depending on the dates present in the metadata these have been aggregated per month or year. The results are shown as a table for each of the time-related metadata per archive. If an aggregation has more than 3 time periods, it is visualised as a bar graph. These tables and graphs are used to allow higher-level insights to be gained more easily. The size of the aggregations, shape of the resulting graphs, the dates present and comparisons between the aggregation are used to gain insight into the archives.

4.2 Results

Metadata dating has been performed on the three archives: the dds server, Kermit and John. First dating was performed on the complete archives. The results are split up by time-related metadata, One section discussing the *access times*, one discussing the *change times* and one discussing the *modify times.*

The dating results are presented as tables showing the number of files in a time period, month or year, dependent on the time range of the archive. For the tables that have more than 3 time periods represented, additional visualisations will be made that help with the interpretation of the results.

Metadata dating can also be performed on a subset of an archive. Metadata dating has been performed on a subset of the Kermit archive, focusing on files related to coding. These results are discussed with regards to the *modify times.*

4.2.1 Access time

The *access times* of all archives can be seen in Table 4.1, Table 4.2 and Table 4.3. The three archives all give approximately similar results. The archives have most of their *access times* in January of 2019. The *access time* shows also another period in which some files were accessed, namely in June 2019.

month	#files
2019-01	124366
2019-06	12977

Table 4.1: Number of files per month by *access time* in the dds server.

month	#files
2019-01	16415
2019-06	437

Table 4.2: Number of files per month by *access time* in the Kermit archive.

month	#files
2019-01	597
2019-06	106

Table 4.3: Number of files per month by *access time* in the John archive.

This can be explained because January is the month in which the course History of Digital Cultures is taught at the University of Amsterdam. The January *access times* are the result of students accessing the files for their research projects. Many students have done queries using the command line tools `find` and `grep`. [Group, 2018d]. These tools access folders and files, altering the *access times* of the files accessed by the students.

The *access times* in June did not have a clear reason for being there. Therefore, a closer look at these files was required. An inspection of the metadata dating results showed that all of these files were either C files or folders.

The C files were the result of a search for a specific piece of C code. The search for C code would also explain the access of the folders that contained the C files. With the folders containing C explained, leaves the other folders that did not contain C code to be explained.

A closer inspection of the *access times* of the folders showed that these were accessed on the same day the MetadataDating tool was run. These *access times* were, in fact, the result of the MetadataDating tool accessing the folders. This is a necessary act to access the time-related metadata of the files and folders within, fully explaining the results.

4.2.2 Change time

The *change times* of all archives can be seen in Table 4.4, Table 4.5 and Table 4.6. The three archives all follow similar results. The *change times* are all after the Grave Diggers Party for the three archives.

month	#files
2017-01	137331
2017-04	9

Table 4.4: Number of files per month by *change time* in the dds server.

month	#files
2018-01	16848

Table 4.5: Number of files per month by *change time* in the Kermit archive.

month	#files
2018-01	703

Table 4.6: Number of files per month by *change time* in the John archive.

It can be seen that the *change times* have not been preserved. The *change times* not being preserved is to be expected because of the nature of the archives. The archives are saved as folder structures instead of images of the original system. By not storing the archives as system images their position changes in the transfer of storage medium, a metadata change. This results in an updated *change time.* Still, the *change times* are after the Grave Diggers Party was held. The later *change times* have been the result of permissions being changed on the archives. There are several reasons that these permissions were changed.

The permission changes were necessary to grant read access to students for their research and restrict their write access to ensure they did not change the contents of the files themselves. Additionally, permission changes were necessary to ensure that only those who were allowed to access the files could do so. In this way the dating results are explained.

The *change times* serve to mark the final alterations to the file's metadata. After these dates, no alterations were made to the metadata of the archives.

4.2.3 Modify time

The *modify time* shows the greatest differences between the archives and gives the most opportunity to discuss the findings. Unlike the previous sections, the results of each archive will be discussed separately.

4.2.3.1 The dds archive

The dds server's *modify times* can be seen in Table 4.7, ranging between 2015 and 2017. 2015, when the most files were modified, is the year in which the files were placed on the HDC server and made available for student research. Two hypotheses could explain the *modify times*. (1) The files were transferred to the server in such a manner that they did not preserve the *modify time* or (2) the FREEZE was carried out in such a manner that it did not preserve the time-related metadata.

The files modified in 2016 and 2017 represent other times when the files have been modified in some way. These events can be separated by the time the file was modified. The separation by *modify times* separates the files by events that caused the modification. Looking at the location and types of files in an event could give some insight into the type of event that caused the modification.

year	#files
2015	129976
2016	7358
2017	9

Table 4.7: Number of files per year by *modify time* in the dds server.

These results are rather disappointing, as they provide no historical insight into the dataset. The only sensible conclusion that it was not preserved well. More research is required to find the cause.

4.2.3.2 The dds archive rerun

The previous results of the dds server were reason for a deeper dive into an older source of the archive in the hope of retrieving time-related metadata that had not been compromised. Thankfully the original extraction of the tar gzip archive file from the FREEZE had been preserved and stored separately. The MetadataDating tool had to be rerun on the older extracted dds server.

The dating results for the *MTimes* for the older copy of the dds server are shown in Table 4.8 and Figure 4.1. The results for the *ATimes* and *CTimes* were similar to those of the dds server used by students, no new insights could be gained there, therefore those results are omitted.

It can be seen that several years stand out in the results of the dds server.

Firstly, the impossible ones.

1902, 1904 and 1958 are all before the Unix epoch in 1970. Meaning that these files somehow have a negative value for their *modify time*.

2030 being in the future would also be quite an impossible feat.

The dates that represent the time that DDS was active are 1994, 1995 and 1996. These are the plausible dating results.

We see a massive spike in 1994, dropping back off in 1995 and very few files in 1996.

The low amount of files in 1996 is to be expected because the FREEZE took place on January 16th of 1996.

year	#files
1902	1
1904	49
1958	4
1980	2
1986	16
1987	28
1988	37
1989	362
1990	361
1991	4714
1992	10872
1993	8454
1994	100438
1995	8818
1996	1239
2014	1677
2030	271

Table 4.8: Number of files per year by *modify time* in the dds server.

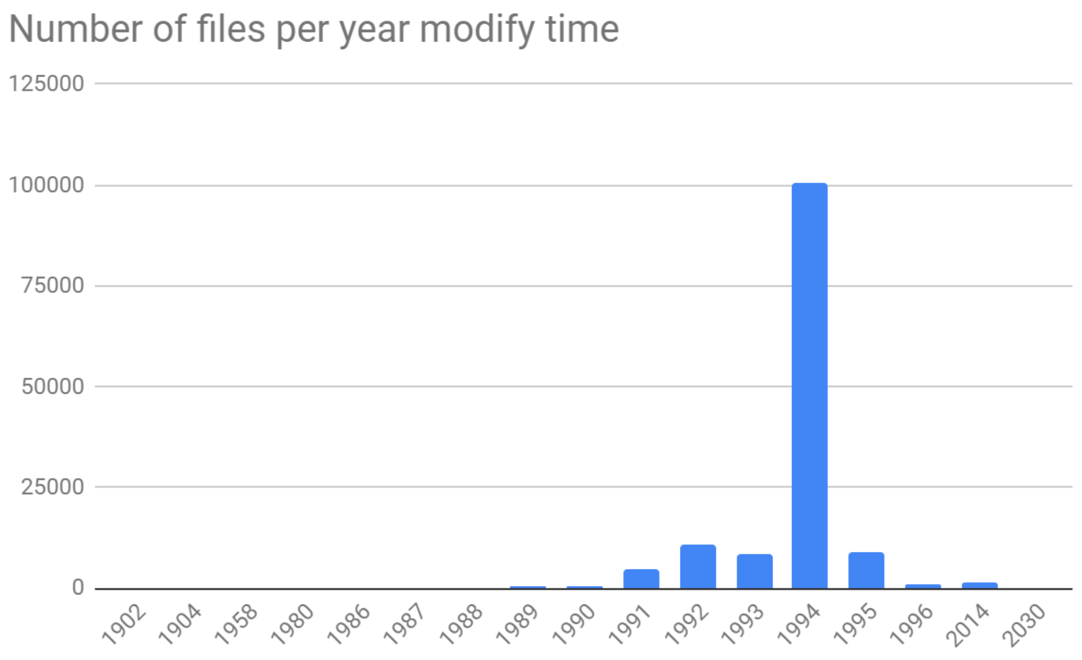


Figure 4.1: Number of files per year by *modify time* in the dds server rerun.

4.2.3.3 The Kermit archive

The Kermit archive is a collection of DDS related files from many different sources. The Kermit archive shows what people wanted to personally preserve of DDS and when. The *modify times* of the files in the Kermit archive represent the final interactions of the creators of the Kermit archive with its contents.

The Kermit archive's *modify times* shown in in Table 4.9 and Figure 4.2, range from 1994 to 2011 and a single file in 2016. The single 2016 result is the folder containing the archive. The 2016 folder was created to contain the archive on the central server. The *modify times* show that the *modify times* have not been compromised by the Grave Diggers Party nor any other actions taken afterwards.

Large spikes in *modify times* appeared in 1998 and 2001. The spikes are when people stopped modifying these files. The spikes could indicate major internal events coming to a close in those years. An internal event indicates that the files have fulfilled their purpose and their development has ceased.

Alternatively, another event could have caused people to add files to their personal archives or stop updating files in their archives in later years. The files themselves could have remained active outside of the Kermit archive.

A closer inspection of the files of 1998 could allow one to test what the event was that caused the spike.

The spike in 2001 is very much expected. When taking a look at the literature on the decline of DDS, 2001 is known as the year that DDS stopped being the platform it once was. The large 2001 spike is the result of the final edits made, the last files being saved for personal archives.

A small uptick in 2011 indicates some renewed interest in DDS. 2011 is the year of the Grave Diggers Party. A manual inspection of the files revealed that these files were windows system files and a folder containing log files related to DDS. The file types suggest that these files were copied in such a manner that their time-related metadata were not preserved, to be added to the archive at the Grave Diggers Party.

Note that the lower amount of files modified in the years before the spikes do not indicate lower activity. Instead, they indicate that files that have seen activity in those years have also seen activity in later years.

year	#files
1994	7
1995	17
1996	325
1997	676
1998	6229
1999	2062
2000	984
2001	4314
2002	2225
2011	12
2016	1

Table 4.9: Number of files per year by *modify time* in the Kermit archive.

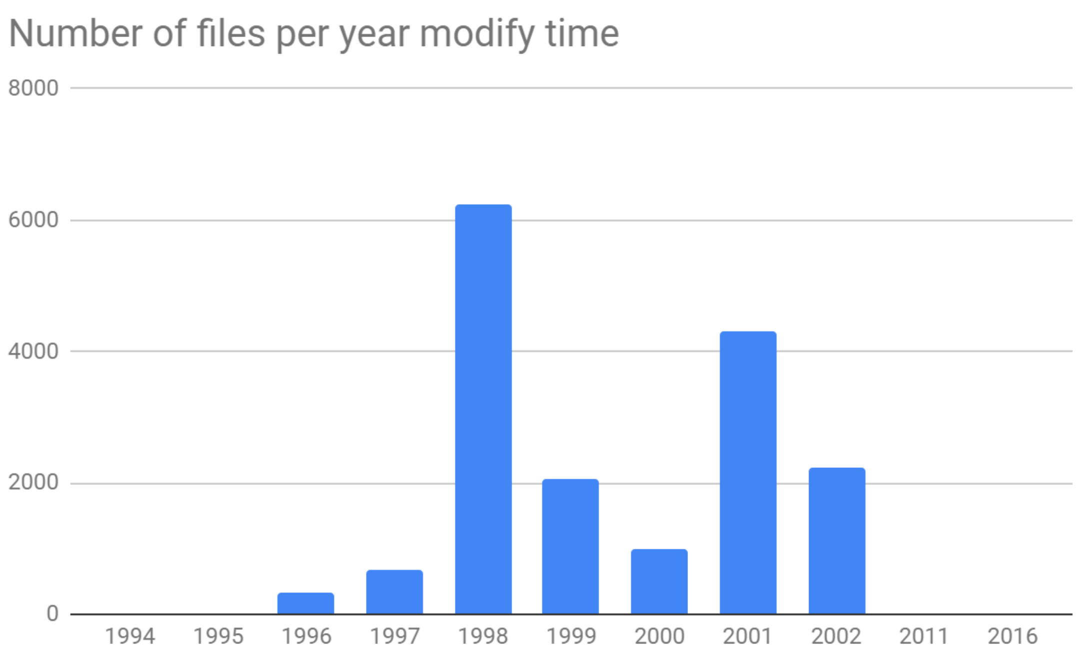


Figure 4.2: Number of files per year by *modify time* in the Kermit archive.

4.2.3.4 The John archive

The John archive consists of backups made from parts of DDS. The results from the John archive can, therefore, be used to estimate the overall development of DDS. The *modify times* of the files in the John archive represent the final modifications made to parts of DDS before it was backed up.

The dating results of the John archive's *modify times* shown in in Table 4.10 and Figure 4.3, range from 1995 to 2016. These dates need to be split into two groups. The first being the results before the Grave Diggers Party in 2011 and those afterards.

The files with a *modify time* after 2011 all have a *modify time* in 2016. A *modify time* in 2016 means that their *modify times* have been altered by some action taken in 2016. 2016 was the year in which the archive was moved to the central server storing all DDS data. The year indicates that part of the archive was not migrated correctly, as the metadata has been altered. For the present research purposes, the 2016 files are disregarded because their *modify times* have been altered after ingestion, invalidating their historical value.

The files before 2011 range between 1995 and 2006. The number of files by *modify time* remains fairly constant until 2001, afterwards, they drop off significantly. A good explanation could be that the date again points to the end of DDS as an online community.

year	#files
1995	83
1996	90
1997	55
1998	121
1999	76
2000	99
2001	79
2002	19
2003	11
2005	1
2006	1
2016	68

Table 4.10: Number of files per year by *modify time* in the John archive.

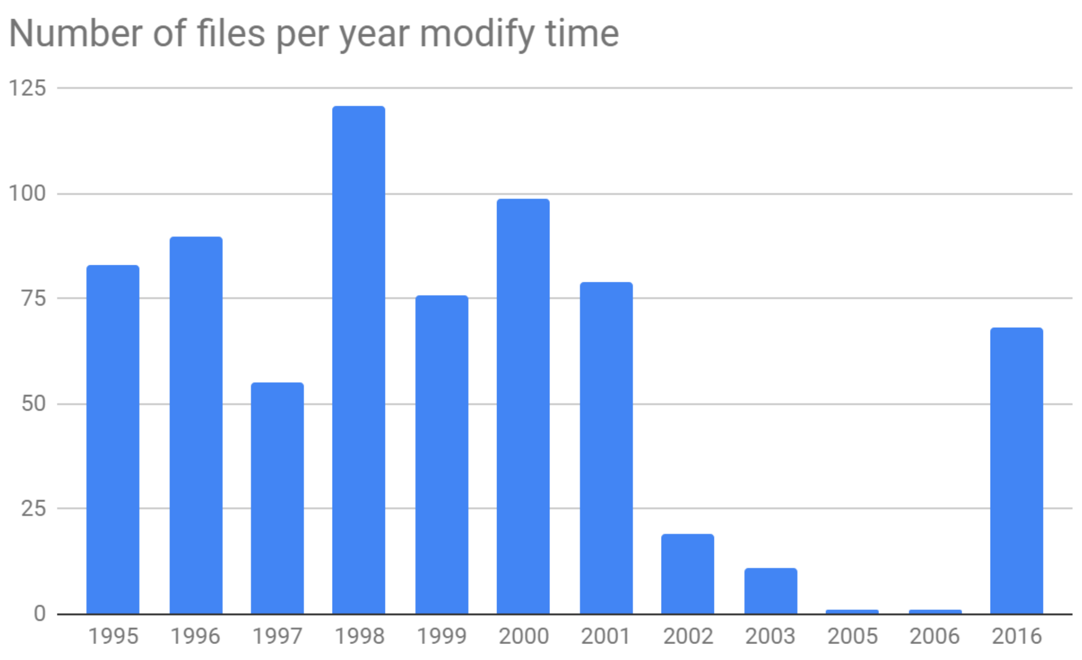


Figure 4.3: Number of files per year by *modify time* in the John archive.

4.2.3.5 Code in Kermit

With metadata dating, it is possible to take a more granular look at a dataset, focusing in on a specific subset of an archive. To illustrate, a closer look has been taken at the *modify times* of code files inside of the Kermit archive. The Kermit archive was selected because it had a large number of files and showed the largest variance between the *modify times* of the files and the code files as a subset.

Extensions seen as code for the comparison in the Kermit archive are sh, c, cgi, o, htm, html and make. These are shell scripts, C source code, CGI scripts, object files, HTML files, also HTML files and makefiles respectively. These are all file types used to code or are closely related to the creation of code.

The results of the code subset will be compared with the results of the Kermit archive overall. Note that the Kermit archive is a collection of DDS related files from many different sources. They were all part of people's personal archives.

The dating results of the Kermit archive as a whole and the code subset can be seen in Table 4.11 and Figure 4.4, separate visualisations for the Kermit archive as a whole and the code subset can be seen in Figure 4.5 and Figure 4.6, respectively. The first thing one notices is that the date range of the code files is smaller. There are no code files with *modify times* as early as 1994, 1995 or 1996. It is important to remember that DDS launched in January 1994. The finding would indicate that people's interest in DDS started out with files other than code files or that the early code was not something that was saved to people's personal archives.

The last files of code have *modify times* in 2002. Here an almost similar drop can be seen in all files in the Kermit archive, with only a few files in 2011 and the containing folder of 2016. The decrease shows people's interest in DDS decline after its fall. These results also show that the files from 2011 are not code files, meaning that the uptick in interest mentioned before, was not related to code files but something else. In future research, a manual inspection of these files could reveal what sparked the interest.

Besides the range of the graphs, the shapes of the graphs also reveal important similarities and differences between the Kermit archive as a whole and the code subset.

1998 is by far the year with the most last *modify times* for both the entire archive and the code subset. The spike indicates that many code and other files that were last modified in 1998 were of such importance that people wished to preserve them. The following two years, 1999 and 2000, show a similar drop in last *modify times* for both code and all files. In 2001 the relative shapes of the graphs are significantly different. While the total amount of files with a last *modify time* of 2001 shows a large spike, the amount of code last modified in 2001 does not share the increase of the overall archive. The difference means that the total amount of non-code files drastically increased. As 2001 is the year of the fall of DDS, the difference could indicate that what people mainly rushed to preserve was not the code of DDS but other files. Where in 2002, after the fall, the relative size differences between code and all files is much smaller, but still slightly favouring other files. These graphs indicate that the users not only had more interest in preserving non-code files over code files but also that the bias for non-code files increased substantially in the later years.

year	#files	#code files
1994	7	0
1995	17	0
1996	325	0
1997	676	5
1998	6229	518
1999	2062	307
2000	984	64
2001	4314	66
2002	2225	159
2011	12	0
2016	1	0

Table 4.11: Number of files and code files per year by *modify time* in the Kermit archive.

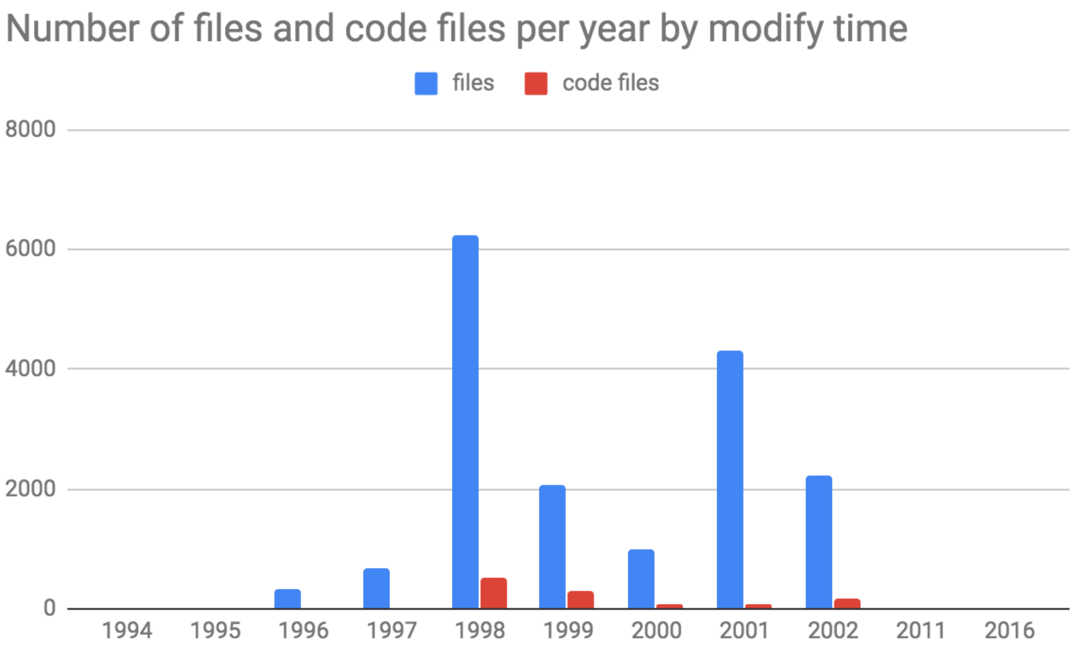


Figure 4.4: Number of files and code files per year by *modify time* in the Kermit archive.

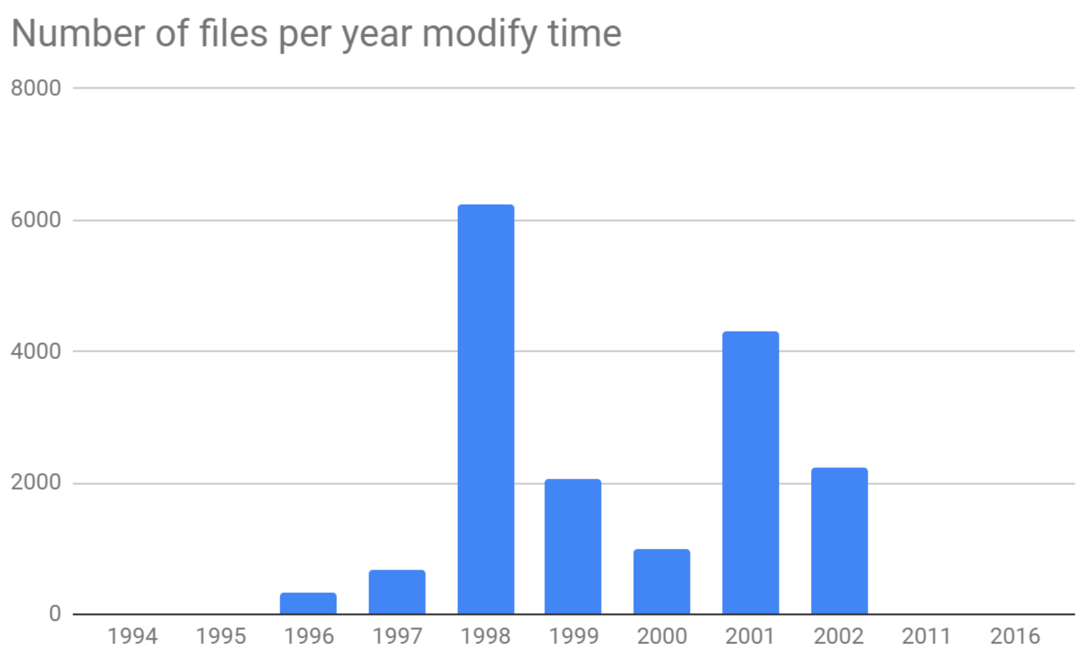


Figure 4.5: Number of files per year by *modify time* in the Kermit archive.

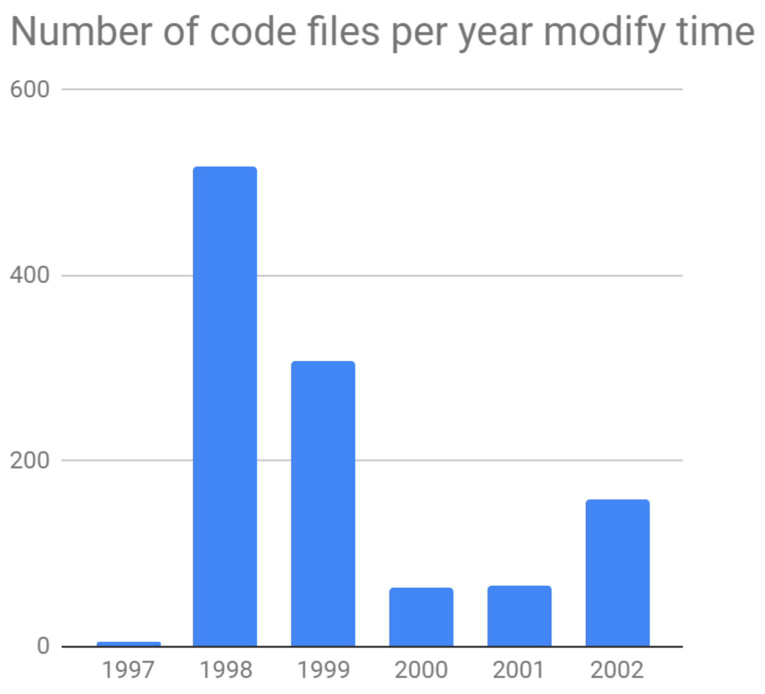


Figure 4.6: Number of code files per year by *modify time* in the Kermit archive.

4.2.4 Key findings

The results from the MetadataDating tool have shown that the integrity of the metadata of the archives has been partially compromised. The dds server used by students only had time-related metadata from after its ingestion, leaving little room for interpretation of the dating results. Thankfully, an earlier copy of the dds server did allow for the retrieval of the *modify times*. The loss of metadata shows the importance of always working with copies and never the original source. As well-intentioned research can compromise the integrity of digital artefacts.

The findings of the *access times* and *change times* further demonstrate the importance of making a good copy of digital artefacts. While *modify times* can be preserved when one is careful, to perform research one needs to access the files, overwriting the *access times* of the files. In Section 3.1.1.5 it has been discussed that the overwriting of *access times* can be turned off.

An important conclusion is that *modify times* and *access times* can be copied over when one is careful with the copying method. However, *change times* cannot! The only way to preserve the change times of one's artefact is making an image of the entire system, preserving the position of the files.

Luckily, the *modify times* in the older copy of the dds server, John and Kermit archives have been preserved, their results allowing for historical interpretation.

The metadata dating results of the archives push forward several dates of interest.

There are several files on the dds server with *modify times* from before the start of DDS.

1994 shows a large spike of files in the dds server.

1995 in the dds server remains unexpectedly low compared to 1994.

1998 shows peaks in both the Kermit and John archives.

2001 shows an uptick for the Kermit archive, while the John archive remains steady in 2001 but sees a drop in files in 2002, where the Kermit archive remains relatively elevated.

2014 is a date present in the dds server that is unexpected.

2016 is a date present in the John archive that is unexpected.

These dates brought forward by the results of metadata dating ask for historical interpretation. Why are these dates different? What events could explain the peaks and valleys in the visualisations of the metadata? To answer these questions one needs to move from the data science questions towards historical questions presented in the next section.

4.3 Historical interpretations

The results of the performed metadata dating give one more insight into the archives than the mere dating results themselves. The results need to be interpreted through their historical context, allowing for higher-level insights to be gained from the archives. In this section, historical insights that have been gained by metadata dating are discussed.

The dds server archive was taken from the FREEZE, making it representative of DDS at the moment the FREEZE was made, in 1996. The results pose questions about the development of the dds server and the DDS as a whole.

DDS started in 1994, but that does not make it unlikely that files older than 1994 existed on the system. The SPARC architecture used by DDS was developed in 1986 and released in 1987, while the operating system used by DDS, Solaris, wasn't released until 1993. It could be possible that the files before and during those dates, but after the Unix epoch, came from different systems or have been updated during the development of the operating system and architecture. A closer inspection of those files could give insight into their purpose, but they are most likely related to the functioning of the operating system.

The difference in *modify times* on the dds server between 1994 and 1995 is surprising. If 1991, 1992 and 1993 are representative for the number of files modified by updates to the operating system, the results would suggest that development of the dds server was almost or entirely exclusive to 1994, the year DDS launched. In future research, a closer inspection of the files modified in 1995 could offer insight into the nature of the files. Were they updates of the operating system or development of DDS?

Is the dds server representative of the other two servers, alibaba and shaman? These servers held other functionalities. It could be possible that new functionalities were developed on those servers while the dds server provided a stable base.

The files with *modify times* in 2014 also pose an interesting question. Are these dates simply wrong like the impossible dates mentioned in the results, or has the archive been altered between its reading in 2013 and its transfer in 2015? This is a question that can only be answered by going back to the original magnetic tapes and re-extracting the archives from there.

Because the John archive consists of backups from DDS one can state that its metadata dating results are indicative for the development of DDS as a whole. From the results of the John archive, one can surmise that the development of DDS was relatively uniform from the beginning of the archive until the decline in 2001. Meaning that there were no major ramp-ups or scale downs in the development speed of DDS when seen on a yearly basis. After 2001 development drops noticeably. But it is not non-existent. That development does not stop after 2001 tells one that even after the stop of free services of DDS in 2001, there were still projects being done in later years.

The John archive has a large group of files with *modify times* in 2016. These files were somehow altered after they were ingested at the Grave Diggers Party in 2011. While the time-related metadata of these files have been altered, their results still serve a role in the interpretation of the results. The size of the group of altered files in 2016 is the error margin for the overall results. In all likelihood, the files in the 2016 group would have been distributed over the years present in the valid results or perhaps also include years close to those of the valid results. However, due to the large amount of files in the group, it could significantly alter the interpretation of the results dependent on their original distribution. The large size means that although the findings based on the John archive are still most likely correct, they do have a known error margin that can skew the interpretation of the results. Additional sources would be required to confidently validate the interpretations. External validation would not be as needed if the error margin were small enough that it could not influence the interpretations.

The Kermit archive, because it is a collection of multiple people's personal archives, gives one an indication of what people wanted to preserve of DDS and how they went about doing that. In contrast with the John archive, the Kermit archive has more peaks and valleys in their *modify times*. The differences in the nature of the archives are important to understand the difference in the visualisation of the *modify times*. Where the John archive is more reflective of the development of DDS, the Kermit archive reflects the final interactions of the users with DDS itself.

The peaks and valleys in the *modify times* of the Kermit archive, as seen in Figure 4.2, indicate a more erratic urge to preserve. The peaks could be linked to events that increased the urge of users to make personal backups of their interactions with DDS. An increased urge to preserve can be clearly seen in 2001 and 2002. The decline of DDS in 2001 was the inciting incident for people to save files for their personal archives.

The peak in 1998 indicates that there was some other event that urged people to add files to their personal archives. 1998 is worth studying due to the volume of files that were last edited in that year. The peak points to a DDS internal event coming to a close or an external event motivating users of DDS to preserve the files that were important to them. The significance of the peak is further validated by a similar, but less pronounced peak in the John archive.

The John and Kermit archives validate the fall of DDS in 2001 as there is a sharp decline of files being created in the John archive after 2001. In the Kermit archive, there is still some activity in people's archives in 2002. The activity in 2002 indicates a delay in people updating their personal archives. Even after the stop of free DDS services being offered in September 2001. The activity in 2002 shows that the creators of the Kermit archive valued the contents enough to pay to access them.

The closer look into the code subset of the Kermit archive gives further insight into what people wanted to preserve in their personal archives. DDS was a website built mostly out of HTML webpages. However, the results show that people were far more inclined to save files other than those related to coding and HTML. Especially during the decline of DDS in 2001 people preserved far more files other than those related to code.

5

Conclusion

Metadata dating has been performed on the three archives related to DDS. The results of the dating, presented in Chapter 4, demonstrate some of the insights that could be gained by performing metadata dating. In this chapter, it will be discussed how the results address the research questions formulated in Section 2.3, the placing of metadata dating as a tool in the field and considerations required when using it, a discussion of new insights that add to the theory of software archaeology and concluding with a future work section.

5.1 The value of aggregated metadata

The goal of the present research is to extract information from born digital artefacts, without requiring specific system knowledge of the artefact itself. To achieve this goal, sub questions have been posed in Section 2.3. This section addresses how each sub-question is answered.

RQ1. Can the time-related metadata of files and folders be extracted?

Yes. The present research demonstrated that if the time-related metadata are present then they can be extracted. In the case of the DDS dataset *access times*, *modify times* and *change times* have been extracted. *Birth time* was not present in the DDS dataset, but could be extracted from different datasets that do contain them with similar means. The extraction of the time-related metadata has been automated with the newly built MetadataDating tool, allowing for easy replication of the research for other datasets.

RQ2. Can the time-related metadata be aggregated in a meaningful way?

The answer is yes. Aggregation of the time-related metadata has been done using the results of the MetadataDating tool and running the results through spreadsheet software. In Chapter 4, the dating resulted in tables and graphs for each of the time-related metadata, showing the number of files within a selected time range for each of the time-related metadata. These aggregations have allowed more significant insights to be gained from the archives as opposed to the results for single files. The insight gained makes the aggregations meaningful.

RQ2.1 Can the aggregations of the time-related metadata be used to check if the metadata of the files are intact?

The dates shown in these representations of the time-related metadata have shown that they allow for easy checks of the integrity of the metadata. Dates outside of the expected ranges can quickly be found at the outer ends of the tables and graphs. This utility makes the aggregation meaningful.

RQ3. How can the aggregations give insight into digital artefacts?

The ultimate goal of metadata dating and the present research is to use the results and their aggregations to gain additional historical insights into the artefacts being studied. The visualisations of the DDS dataset time-related metadata show the chronological order of events in the dataset. There are peaks and valleys in the visualisations and some of these have been explicitly linked to known events that have taken place at those times. These results not only validate existing historical insights into DDS but are also helpful in the discovery of new historical insights around these known events.

Besides known events, previously unknown events have been discovered also. On closer inspection of the files, greater insight could be gained about the nature of the discovered events. The files themselves could point to the events or the dates shown by the metadata could be linked to external events.

In addition to discovering events and placing them chronologically, metadata dating can be used to gain insight into the reason(s) why these events happened. By having these files grouped chronologically, they can be easily inspected chronologically. The inspection can happen on a per-file basis but also from a granularity in between individual files and the system as a whole.

Targeted metadata dating can be used to show differing interest between file types. The difference in interest asks for historical interpretation. For example, why are people more inclined to preserve one type of file over an other?

It has been shown that the aggregation of time-related metadata can be used to gain historical insight into digital artefacts.

Metadata dating provides a lens through which one can study digital artefacts. Without technical knowledge of the artefact, one can view the time-related metadata and draw conclusions based on the findings.

Dates in the results that set themselves apart from the others by having many or few files in their aggregations, peaks and valleys in their visualisations, are reason for further investigation. Without system knowledge, secondary sources can be consulted as to the significance of those dates. Perhaps significant events took place in the world or events related to the artefact itself. With rudimentary system knowledge, the files in the peaks and valleys can be inspected. In conclusion, metadata dating proves itself to be an archaeological tool, a shovel or a duster, unearthing the artefact and showing one just where to look.

5.2 Metadata dating as a tool for software archaeology

Metadata dating can be used as an effective tool in software archaeology. When and how to use metadata dating is important. This section provides some recommendations related to how and when to use metadata dating for software archaeology.

5.2.1 Place in field

Metadata dating as a concept provides a methodology for the study of digital artefacts. The MetadataDating tool is an implementation of the technical aspects required to perform metadata dating. The MetadataDating tool is intended to be used on copies of the original artefact because, as discussed previously, it can alter the metadata of the artefact it is used to study. The role of the tool is after the ingestion of the digital artefact.

Once the artefact is ingested and its data integrity can be guaranteed, tools for the study of digital artefacts can be used on copies of the original artefact. These include all types of tools that serve a digital forensic purpose, such as the MetadataDating tool. These are tools that look at the contents and or metadata of files that make up the digital artefact.

The MetadataDating tool provides the user with a more accessible format of the time-related metadata of their digital artefact. The accessibility MetadataDating tool provides, allows the researcher to gain insight using a data-driven approach without requiring the researcher to possess technical system knowledge of their artefact. It is up to the researcher to use the data presented by the MetadataDating tool and put it into context with additional sources. The MetadataDating tool allows the researcher to use the data science questions the MetadataDating tool answers and use them to answer more historical questions.

The MetadataDating tool answers the call for tools fitting the requirements of historical and archival research.

5.2.2 Considerations when performing metadata dating

Working with any type of heritage, it is always important to take care not to damage the artefact one is working with. The beauty of digital artefacts is that research can be performed on a copy of the original artefact. The results of metadata dating show the importance of doing so.

The research previously performed on the archives had necessitated changes to be made to the metadata of the archives. The previously done research altered the results of the metadata dating. It is possible to go back to the original backups of the data, which should be system images or they lose valuable metadata, make copies of the backups and reperform metadata dating on the copies.

Performing metadata dating on new copies of the system images would result in non-altered dating results. Making new system images was not done in the present research due to time and access to hardware limitations and the following difficulty.

The process of metadata dating itself alters the metadata of folders in the archive. To perform metadata dating on a file, the folder containing the file needs to be accessed, changing the *access time* of the folder. The MetadataDating tool avoids altering the results of metadata dating by first dating the folder and then accessing the folder and dating the files and folders inside. The need to access the containing folders makes it so that metadata dating can only be performed once on a copy. A second attempt would always show the *access time* of the previous metadata dating for folders. Due to the iterative testing required when developing a new tool, it was decided to not perform metadata dating on fresh copies of the backups of the archives. As the process would require frequent careful handling of the backups.

Data integrity is a vital part of working with digital artefacts. Ensuring that the artefact is not compromised is traditionally done with checksums. The checksum approach looks at a file and computes a hash from that file. If the file remains unchanged the hash will always be the same. The checksum approach does not consider the metadata as it is not part of the file itself. Using metadata dating, the integrity of the time-related metadata can be checked. Although the metadata dating process needs to be performed on a copy to maintain the integrity of the original.

5.3 New insights

Working with the MetadataDating tool has given greater insight into the DDS and the theory of performing software archaeology.

A model is proposed for consideration of the authenticity of digital artefacts. The model looks at a digital artefact from the perspective of the files that make up the artefact. The other levels take a step backwards from the files and consider the authenticity from a higher level of abstraction. Using the proposed model it is shown that metadata dating can be used to check the authenticity of a digital artefact from a metadata level, much as checksums can be used to check the authenticity on a file level.

There is an inherent paradox about working with historical artefacts. They are altered by their preservation and study. No matter how slight the interaction with a historical artefact, the artefact is inadvertently changed by it. Take a book for example. Simply moving the book exposes the book to friction with the surface it was on and where it will be put on causing minor damage that becomes visible over time. By reading the book, one exposes it to light. The exposure to light slowly changes the colour of the paper over time.

Digital artefacts are very similar to their analogue counterparts in this regard. To ingest a digital artefact, the storage medium it is placed on (such as a hard drive or USB stick) needs to be read. The reading can change the metadata on the storage medium. Once ingested and working copies have been made, each interaction with a working copy will change the metadata of that working copy. The changes to the metadata by each interaction is a retrieval paradox. Metadata dating shows the retrieval paradox in its utilisation. By retrieving the *access times* of the files and folders of a digital artefact it will change the *access times* of the folders of the artefact. The same will be true for any tool that looks inside files, as it will change the *access time* of the files studied.

The present research has demonstrated that the order a researcher works in is vitally important. Irreparable damage can be caused to the metadata of a digital artefact by merely looking at the files and folders. The importance of always working with copies of copies cannot be emphasised enough.

Due to the nature of metadata and the retrieval paradox, metadata dating should be the first action a researcher performs on their working copy. There is only one chance to perform metadata dating. Once any folder or file is opened in the artefact, their *access times* are overwritten. To perform metadata dating a new copy needs to be made that does not have its metadata altered by a previous interaction.

5.4 Future work

In this section avenues for future research are given based on the results and findings of the present research.

5.4.1 Dataset limitations

In the data science field, a researcher can choose the data set that fits their method. In software archaeology, it is needed to choose a method that fits one's dataset.

In Chapter 4, new insights have been gained from the *modify times* of the researched archives. These new insights demonstrate the value of metadata dating and the Metadata-Dating tool.

Although the *access times* and *change times* did prove their value as a check on the integrity of the metadata, it would be preffered to also perform metadata dating on datasets with these values intact, further demonstrating the power of these methods.

To perform metadata dating on a dataset with its *access times* intact, it is vital that the digital artefact is copied carefully, keeping the *access times* unaltered. Metadata dating can then be performed on a copy, but only once as it will alter the *access times* of the folders part of the artefact.

Metadata dating also needs to be performed on a dataset with its *change times* intact. *Change times* are less volatile than the *access times* and so have more than one opportunity to be performed. It is, however, key that no permission, name, location or other metadata changes can be performed before metadata dating is performed. These changes can be required to perform other research, it is, therefore, advisable to first perform metadata dating prior to performing other research that could require metadata alterations.

The DDS dataset did not have *birth times*. The lack of *birth times* is due to the filesystem used in the storage of the dataset. Note that the *birth times* were not lost, they were simply never present in the original artefacts. There are file systems that do possess *birth times*. To show the full capabilities of metadata dating, dating needs to be performed on a dataset that does have *birth times* present, to showcase the insights that could be gained using *birth times*.

If metadata dating could be performed on a system that has multiple of the time-related metadata intact, interactions between the visualisations of the time-related metadata could be shown.

5.4.2 MetadataDating tool

The current implementation of the MetadataDating tool is compatible with Ubuntu-based Linux distributions. It uses the default implementation of `find` in that distribution. Other distributions can use different implementations, making the MetadataDating tool non-compatible with those distributions. MacOS also has a different implementation of `find` making the MetadataDating tool incompatible. Three options present themselves to deal with the incompatibility on different systems.

The first is to make a different version of the MetadataDating tool that is compatible with macOS and hopefully other distributions. Other non-compatible distributions will require their own versions or must go without.

The second option is to create a more robust version of metadata dating that can work on more systems. A more robust version will have the benefit of having only one version of the tool to maintain. The downside is that a more robust version will most likely need to be built in a programming language that is universally available on all or at least most common systems. The language would need to have the required functionalities to perform metadata dating while working on a fast array of different operating systems and file systems. The language requirements could lead to other compatibility issues and increase the time required to run the tool.

A third option would be to virtualise the functionality of the MetadataDating tool. Other comparable tools such as ‘Count lines of code’ and ‘Carbon dating the web’ offer virtualisation in the form of a Docker container. Virtualisation would allow the MetadataDating tool to be used on any system that has Docker installed.

At the time of writing the `stat` command was chosen for the promised functionality of providing the *birth time*. This is a functionality is provides on some distributions, such as FreeBSD, but it turns out that This functionality is blocked on The GNU Coreutils stat command due to lacking kernel support of the xstat() interface [Blake, 2011; pixelbeat.org, 2012]. This results in the *birth times* always returning an empty value on Ubuntu. Note that this does not invalidate the *birth time* results, as there is no birth time present in the dataset. There is, however, a need for a more robust replacement for retrieving the *birth time* in datasets do have this metadata present.

Hopefully POSIX will add birth times to their standard that it will be universally supported by compatible operating systems. For now different options should be explored, such as using using the `debugfs` command [Moiseev, 2015].

5.4.3 Levels of authenticity

In Section 2.2, a model for the preservation of digital artefacts is discussed. This paper takes the approach of taking a file as the base level, discussing preservation from a file level and then moving to lower levels of abstraction all the way to a hardware implementation level.

Taking the file level as a basis is, however, not the only way to look at the preservation of digital artefacts. A reverse approach can be taken, discussing preservation from the hardware implementation level and moving to higher levels of abstraction.

Additionally, software needs not be seen from a hardware perspective at all. The hardware level can be taken out, leaving the functional level as the highest level of preservation.

Different views can be taken on the functional level, that could lead to widely different views on authenticity.

Unlike data integrity, which can be objectively determined, the authenticity of a digital artefact is more complex. When speaking of authenticity from any other perspective than 100% original hardware and software (meaning the actual machines the digital artefact worked on with the actual hardware interfaces: Monitor, keyboard and the like), trade-offs need to be made.

Moving to different hardware poses questions about the type of hardware. How does the move from one hard drive to another affect the experience? What if the data is moved to an SSD instead of a hard drive?

Not only hardware can be changed, but the software can also be altered or replaced. How does updating the software of the artefact alter the experience? Or the move from one type of software to another? How important is it to have authentic software if the functionality is the same?

An important consideration is also how alterations in hardware and software can influence performance as opposed to mere functionality. An argument can be made that to truly experience the internet of the ‘90s, pictures should be loaded in slowly line by line. As was the case, due to slower internet speeds. Replicas and web archives of websites in the early ‘90s often lack the slower loading in of a picture due to faster internet speeds experienced today. Fundamentally altering the experience and thus the authenticity.

A more comprehensive overview is required on the authenticity and preservation of digital artefacts. There are different perspectives one can take concerning the authenticity and preservation of digital artefacts. Depending on the goal of the researcher or archaeologist a different view might be appropriate and even more affordable, to assist them to select the right tool for their preservation efforts.

References

Gerard Alberts, Marc Went, and Robert Jansma. Archaeology of the Amsterdam digital city; why digital data are dynamic and should be treated accordingly. *Internet Histories*, 1(1-2):146–159, 2017. 2, 11, 14

apple.stackexchange.com. How to keep original “date created” when copying via Terminal?, 2015. URL `https://apple.stackexchange.com/questions/199536/how-to-keep-original-date-created-when-copying-via-terminal`. Accessed: 2020-01. 38

Grant Atkins. Carbon Dating the Web, version 4.0, 2017. URL `https://ws-dl.blogspot.com/2017/09/2017-09-19-carbon-dating-web-version-40.html`. Accessed: 2020-01. 21

Eric Blake. Re: How to find by birth time?, 2011. URL `https://lists.gnu.org/archive/html/bug-findutils/2011-11/msg00015.html`. Accessed: 2020-06. 47, 76

John D Blischak, Emily R Davenport, and Greg Wilson. A quick introduction to version control with Git and GitHub. *PLoS computational biology*, 12(1), 2016. 22

Stewart Brand. Escaping the Digital Dark Age. *Library Journal*, 124(2):46–48, 1999. 12

Niels Brügger. Website history and the website as an object of study. *New Media & Society*, 11(1-2):115–132, 2009. 1

Niels Brügger. Digital Humanities in the 21st Century: Digital Material as a Driving Force. *DHQ: Digital Humanities Quarterly*, 10(3), 2016. 13

Niels Brügger, Ian Milligan, Anat Ben-David, Sophie Gebeil, Federico Nanni, Richard Rogers, William J Turkel, Matthew S Weber, and Peter Webster. Internet histories and computational methods: a “round-doc” discussion. *Internet Histories*, 3(3-4):202–222, 2019. 17

Digital Preservation Coalition. 'the Digital City Revives: A Case Study Of Web Archaeology', 2016. URL https://www.dpconline.org/events/digital-preservation-awards/the-digital-city. Accessed: 2020-01. 5

A Crymble. Historians are becoming computer science customers postscript. *Personal website*, 2015. URL https://ihrdighist.blogs.sas.ac.uk/2015/06/historians-are-becoming-computer-science-customers-postscript/. 17

Ward Cunningham, Andrew Hunt, Brian Marick, and Dave Thomas. OOPSLA 2001 Software Archeology Workshop, 2001. URL https://web.archive.org/web/20060616052627/http://visibleworkings.com/archeology/. Accessed: 2020-01. 13

Patrick Daly and Thomas L Evans. *Digital archaeology: bridging method and theory*. Routledge, 2004. 12

DDS. Nieuwe toekomst voor De Digitale Stad, 2001. URL https://web.archive.org/web/20010421115727/http://pers.dds.nl/?1718. Accessed: 2020-01. 4

Tjarda De Haan. GEVONDEN: FREEZE! Unieke webarcheologische vondst, 2014a. URL https://hart.amsterdam/nl/page/37138/gevonden-freeze. Accessed: 2020-01. 4

Tjarda De Haan. Grave Diggers Party: a party with a cause — Hart Amsterdammuseum, 2014b. URL https://hart.amsterdam/nl/page/34382/grave-diggers-party-a-party-with-a-cause. Accessed: 2020-01. 4

Tjarda De Haan. DE DIGITALE STAD GEKRAAKT— Hart Amsterdammuseum, 2016a. URL https://hart.amsterdam/nl/page/55870. Accessed: 2020-01. 5

Tjarda De Haan. PRESENTATIES DE DIGITALE STAD GEKRAAKT. Unieke webarcheologische opgravingen, 2016b. URL https://hart.amsterdam/nl/page/56068/presentaties-de-digitale-stad-gekraakt. Accessed: 2020-01. 5, 26

Tjarda De Haan. Klaar om te graven!, 2016c. URL https://hart.amsterdam/nl/page/55422/klaar-om-te-graven. Accessed: 2020-01. 27

Tjarda De Haan. Project" The Digital City Revives" A Case Study of Web Archaeology. In *iPRES*, 2016d. 4

Tjarda de Haan, Robert Jansma, and Paul Vogel. *DIY Handboek voor Webarcheologie. Do It Yourself: Plan, graaf, reconstrueer en ontsluit!* Amsterdam Museum, 2017. URL https://hart.amsterdam/image/2017/11/17/20171116_freeze_diy_handboek.pdf. 5, 8, 25

Andrew Forward and Timothy C Lethbridge. The relevance of software documentation, tools and technologies: a survey. In *Proceedings of the 2002 ACM symposium on Document engineering*, pages 26–33, 2002. 11

Free Software Foundation. Bash, 2017. URL `https://www.gnu.org/software/bash/`. Accessed: 2020-01. 46

Matthew Garrett. Powering down. *Communications of the ACM*, 51(9):42–46, 2008. 38

GNU. File timestamps (GNU Coreutils), n.d.a. URL `https://www.gnu.org/software/coreutils/manual/html_node/File-timestamps.html`. Accessed: 2020-01. 36, 38

GNU. mv invocation GNU Coreutils, n.d.b. URL `https://www.gnu.org/software/coreutils/manual/html_node/mv-invocation.html`. Accessed: 2020-01. 37

GNU. touch invocation (GNU Coreutils), n.d.c. URL `https://www.gnu.org/software/coreutils/manual/html_node/touch-invocation.html`. Accessed: 2020-01. 36

KM Goh. Carbon dating. *Carbon Isotope Techniques*, 1:125, 1991. 8, 32

Michael A Goldstein, Pavitra Kumar, and Frank C Graves. Computerized and high-frequency trading. *Financial Review*, 49(2):177–202, 2014. 38

The Open Group. POSIX.1-2017, 2018a. URL `https://pubs.opengroup.org/onlinepubs/9699919799/`. Accessed: 2020-01. 46

The Open Group. POSIX.1-2017 find, 2018b. URL `https://pubs.opengroup.org/onlinepubs/9699919799/utilities/find.html`. Accessed: 2020-01. 46

The Open Group. POSIX.1-2017 General Concepts, 2018c. URL `https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_16`. Accessed: 2020-01. 33

The Open Group. POSIX.1-2017 grep, 2018d. URL `https://pubs.opengroup.org/onlinepubs/9699919799/utilities/grep.html`. Accessed: 2020-01. 52

Andy Hunt and Dave Thomas. Software archaeology. *IEEE Software*, 19(2):20–22, 2002. 13

Simon Kincaid. After the fire: reconstruction following destructive fires in historic buildings. *The Historic Environment: Policy & Practice*, 11(1):21–39, 2020. 12

Christopher A Lee, Matthew Kirschenbaum, Alexandra Chassanoff, Porter Olsen, and Kam Woods. Bitcurator: tools and techniques for digital forensics in collecting institutions. *D-Lib Magazine*, 18(5/6):14–21, 2012. 18

Rüdiger Lincke, Jonas Lundberg, and Welf Löwe. Comparing software metrics tools. In *Proceedings of the 2008 international symposium on Software testing and analysis*, pages 131–142, 2008. 6

Neil Matthew and Richard Stones. *Beginning linux programming.* John Wiley & Sons, 2008. 33

Microsoft. Performance data from Average Disk sec/Read counter of the PhysicalDisk performance object | Microsoft Docs, 2014. URL `https://docs.microsoft.com/en-us/previous-versions/office/exchange-server-analyzer/aa996220(v=exchg.80)`. Accessed: 2020-01. 38

Igor Moiseev. Get file creation time on Linux with EXT4, 2015. URL `http://moiseevigor.github.io/software/2015/01/30/get-file-creation-time-on-linux-with-ext4/`. Accessed: 2020-06. 76

Janne Nielsen. Experimenting with computational methods for large-scale studies of tracking technologies in web archives. *Internet Histories*, 3(3-4):293–315, 2019. 17

NVIDIA. G-SYNC | NVIDIA Developer, n.d.a. URL `https://developer.nvidia.com/g-sync`. Accessed: 2020-01. 13

NVIDIA. NVIDIA HairWorks | NVIDIA Developer, n.d.b. URL `https://developer.nvidia.com/hairworks`. Accessed: 2020-01. 13

University of Amsterdam. History of Digital Cultures, 2014. URL `https://studiegids.uva.nl/xmlpages/page/2014-2015-en/search-course/course/14066`. Accessed: 2020-01. 5

University of Amsterdam. History of Digital Cultures, 2015. URL `https://studiegids.uva.nl/xmlpages/page/2015-2016-en/search-course/course/21188`. Accessed: 2020-01. 5

pixelbeat.org. Coreutils Inbox, 2012. URL `http://www.pixelbeat.org/patches/coreutils/inbox_aug_2012.html`. Accessed: 2020-06. 47, 76

Tara Raj. What's new in WSL in Windows 10 Fall Creators Update, 2017. URL `https://devblogs.microsoft.com/commandline/whats-new-in-wsl-in-windows-10-fall-creators-update/`. Accessed: 2020-01. 46

Gordana Rakic and Zoran Budimac. Problems in systematic application of software metrics and possible solution. *arXiv preprint arXiv:1311.3852*, 2013. 6

Patrice Riemens and Geert Lovink. Local networks: digital city Amsterdam. *Global networks, linked cities*, pages 327–45, 2002. 4

Gregorio Robles, Jesus M Gonzalez-Barahona, and Israel Herraiz. An empirical approach to Software Archaeology. In *Proc. of 21st Int. Conf. on Software Maintenance (ICSM 2005), Budapest, Hungary*, pages 47–50, 2005. 13

Els Rommes, Ellen van Oost, and Nelly Oudshoorn. Gender in the design of the digital city of Amsterdam. *Information, Communication & Society*, 2(4):476–495, 1999. 3

ReindeR Rustema. The Rise and Fall of DDS: evaluating the ambitions of Amsterdam's Digital City. *Unpublished master's thesis). University of Amsterdam, Amsterdam.*, 2001. URL `https://reinder.rustema.nl/dds/rise_and_fall_dds.pdf`. 4

Hany M SalahEldeen and Michael L Nelson. Carbon dating the web: estimating the age of web resources. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1075–1082, 2013. 21

F Schmuck, R Haskin, and A GPFS. Shared-Disk File System for Large Computing Clusters. In *Int. Conf. on File and Storage Technologies (FAST)*, 2002. 38

Hankeun Son, Seongjin Lee, Gyeongyeol Choi, and Youjip Won. Coarse-grained mtime update for better fsync () performance. In *Proceedings of the Symposium on Applied Computing*, pages 1534–1541, 2017. 38

StephaneG31. .net projects and .config files · issue #374 · AlDanial/cloc · GitHub, 2019. URL `https://github.com/AlDanial/cloc/issues/374`. Accessed: 2020-06. 19

Kees Teszelszky. Web archaeology in The Netherlands: the selection and harvest of the Dutch web incunables of provider Euronet (1994–2000). *Internet Histories*, 3(2):180–194, 2019. 1, 3

Stuart G Tucker. Emulation of large systems. *Communications of the ACM*, 8(12):753–761, 1965. 27

Ask Ubuntu. How do i find the creation time of a file?, 2014. URL https://askubuntu.com/questions/470134/how-do-i-find-the-creation-time-of-a-file. Accessed: 2020-01. 34

Ask Ubuntu. When is Birth Date for a file actually used?, 2017. URL https://askubuntu.com/questions/918300/when-is-birth-date-for-a-file-actually-used. Accessed: 2020-01. 34

unixtutorial.org. atime, ctime and mtime in Unix filesystems, 2008. URL https://www.unixtutorial.org/atime-ctime-mtime-in-unix-filesystems/. Accessed: 2020-01. 33

Judith van Gent. Digital Preservation Award gewonnen! — Hart Amsterdam museum, 2016. URL https://hart.amsterdam/nl/page/121992/digital-preservation-award-gewonnen. Accessed: 2020-01. 5

Waag. Archeologische Dienst DDS, 2014. URL https://waag.org/nl/project/archeologische-dienst-dds. Accessed: 2020-01. 4

Marc Went. THE QUEST FOR THE DDS AVATARS. A webarchaeological discovery, 2015. URL https://hart.amsterdam/nl/page/49413/the-quest-for-the-dds-avatars. Accessed: 2020-01. 5