Towards a methodology for software preservation

Brian Matthews(1), Arif Shaon(1), Juan Bicarregui(1), Catherine Jones (1) , Jim Woodcock (2), Esther Conway (1)

(1) STFC Rutherford Appleton Laboratory, Chilton, Didcot, OXON, OX11 0QX, UK

(2) Department of Computer Science, University of York, Heslington, York, YO10 5DD, UK.

brian.matthews@stfc.ac.uk

Abstract

Only a small part of the research which has been carried out to date on the preservation of digital objects has looked specifically at the preservation of software. This is because the preservation of software has been seen as a less urgent problem than the preservation of other digital objects, and also the complexity of software artefacts makes the problem of preserving them a daunting one. Nevertheless, there are good reasons to want to preserve software. In this paper we consider some of the motivations behind software preservation, based on an analysis of software preservation practice. We then go on to consider what it means to preserve software, discussing preservation approaches, and developing a performance model which determines how the adequacy of the a software preservation method. Finally we discuss some implications for preservation analysis for the case of software artefacts.

Introduction

Software is a class of electronic object which is by its very nature digital and which is often a vital pre-requisite to the preservation of other electronic objects. However, software has many characteristics which make its preservation substantially more challenging than that of many other types of digital object. Software is inherently complex, normally composed of a very large number of highly interdependent components and often forbiddingly opaque for people especially those who were not directly involved in its development. Software is also highly sensitivity to its operating environment as a typical software artefact has a large number of other items upon which it depends including compilers, runtime enviroments, operating systems, documentation, and even hardware platform with its built in software stack. So preserving a piece of software may involve preserving much of the context as well.

Handling these challenges is therefore a major barrier to the preservation of software. So much so that often, the preservation of software is seen as a secondary activity, less critical that the preservation of the data it manipulates. However, in many cases, such data is uninterpretable without the software to handle it; and recreating software from partial information can be a near impossible task.

Software preservation is thus a relatively underexplored topic and there is little practical experience in the field of software preservation as such. The results reported in this paper arose from a UK JISC sponsored study into the significant properties of software for preservation[1], and subsequently in a JISC project into methods and tools for software preservation[2]. Given the relative immaturity of the field, the studies became an exploration of the notion of software preservation, considering the stakeholders and motivations behind software preservation as much as identifying the methods and technology required to support it.

Different communities have needs to preserve software, such as libraries and archives, managers of data archives which have a need to preserve associated software, and software developers themselves maintaining and reusing software over the long term. We consider different approaches to software preservation which vary from an emphasis on preserving the software executables directly, which uses hardware preservation and emulation, to an emphasis on preserving the essential behaviour of software in a new context via migration and porting.

We discuss some concepts useful for a software preservation methodology, considering the stages of *retrieval, reconstruction and replay* which need to be passed through to reproduce a usable performance of a software product. We identify a notion of *adequacy* of preservation, an aspect of the authenticity of preservation which tests the future performance of software against specified preservation properties once it has been reconstructed into a working version in the new environment. This notion allows us to relate our approach to software preservation to concepts from the OAIS information model; indeed the approach can be seen as a specialisation of OAIS for the case of software. We comment on using an existing preservation methodology for software and finally discuss some experiences of using the approach in practice.

[1] Joint Information Systems Committee (JISC) study into the *Significant Properties of Software* (2007).

[2] Joint Information Systems Committee (JISC) sponsored project *Tools and Guidelines for Preserving and Accessing Software Research Outputs* (2007-09).

Why Preserve Software?

A key question to consider is why might it useful do preserve software. After all, software has a track record of being both being very fragile and very disposable.

Software is *fragile* as it is very sensitive to changes in environment: hardware, operating system, versions of systems (e.g. programming languages and compilers) and configuration. When the environment changes, software notoriously stops working, crashes losing vital data, or works but not as originally intended, with missing or differing functionality. The last case can be particularly damaging, as the software may seem to work but produces subtly different results. For example, compiling with a different floating point module may produce quite different results in the analysis. The complexity of the software makes it difficult to make the required adjustments so that is functions correctly in the new environment.

Software is *disposable* as in the face of environment change and the complexity of large-scale systems, developers often throw away previous software and start again from scratch. If the data has been preserved, it may be easier to write new software rather than wrestle with legacy, and you may be able to produce a faster, more user-friendly system which operates in a modern environment.

Together, these make the preservation of software appear both difficult and unnecessary. However, there are also good reasons to preserve software, especially in a research and teaching environment. Some of these reasons are as follows.

Museums and Archives

A small but significant constituency of software preservation is those museums, archives, and enthusiasts which are interested in preserving aspects of the history of computing. These institutions wish to preserve important software artefacts as they were at the time of their creation or use, so that future generations can study and appreciate the computers available that particular period, and trace their development.

Some museums concentrate on preserving hardware, with machines are often kept in working operation, so there is a need to preserve the software to demonstrate the function of the machine. Others archives are interested in preserving the software alone, typically via a web presence. For example, the Multics History Project[3] is a effort to locate and engage the original experts on the Multics operating system to capture their knowledge before they die.

There has been given some consideration of how to preserve software in this context (Zabolitsky 2002). However, this is largely limited to preserving historic software with the historic hardware, so the major concern is preserving the code on some physical media, with appropriate backup and replication strategies; these preservation actions are similar to those for other digital artefacts. The problem of preserving the *usage* of the software in a future context is not considered in detail.

Preserve a complete record of work

Software is frequently an output of research. This is particularly the case in Computer Science where the software itself is an important test of the hypothesis of the research. However, software as an output of research extends beyond Computer Science as many research projects across all disciplines now frequently have an aspect of computing and programming .

If university archives and libraries are going to maintain a complete record of research, then the software itself should be preserved. Frequently, theses include code listings or CD-ROM's of the supporting software. However, while the theses are stored on library shelves, software is not necessarily preserved against media change or change in the computing environment making the code difficult to run. Research projects again frequently produce software, to support their claims, so the results of the project are hard to interpret and evaluate without the software. However, at the end of the project, unless the software is taken up in a subsequent project, there is little incentive or resources to maintain access to the software in a usable form. Library preservation strategies thus need to accommodate the preservation of software as well as other research outputs.

Preserving the data

In order to verify the claims of a research project, then they should be reproducible from its data. It may be enough to rerun the analysis on current software if the original data has been preserved. But in other circumstances, for example to test accuracy or detect fraud, there may be a need to rerun the original software precisely to reproduce the exact result so they can be judged on the results as they saw them at the time. Newer software may have errors corrected, have higher performance or accuracy characteristics, or else have improved analysis algorithms or visualisation tools. All these factors may lead later analysis of the data to different conclusions but the scientists should nevertheless be judged on the view they were able to take at the time.

Further, data which is collected on sophisticated equipment or facilities is expensive; other data which is recording specific events is non-reproducible. In these cases, it is desirable to preserve and reuse the data to maximise its scientific potential, and it is often necessary to also preserve some supporting software to process the data format, and to provide data analysis. This is also relevant to the preservation of other digital objects. Preservation of document or image formats requires the preservation of format processing and rendering software in order to keep the content accessible to future users.

Thus software also needs to be preserved to support the preservation of data and documents, to keep them live and reusable. In this case, the prime purpose of the preservation is not to preserve the software in itself, so it may be suitable not to ensure that that software is reproduced in its exact form, but only sufficiently well preserved to process the target data accurately.

Handling Legacy

Perhaps the prime motivation to preserve software for

[3] http://www.multicians.org/mhp.html

many organisations is to save effort in recoding. It is frequently seen as more efficient to reuse old code, or keep old code running in the face of software environment change than to recode. This is certainly the reason for the maintenance of most existing software repositories, and forms a significant part of the effort which is undertaken by software developers. Handling legacy software is usually seen as a problem, and many strategies are undertaken in order to rationalise the process, to make it more systematic and more efficient Thus the best practice on *software maintenance and reuse*, a long recognised part of good software engineering, also supports good software preservation. If you can find an existing package or library routine, why bother rewriting it? Of course in these circumstances you need assurance that the software will run in your current environment and provide the correct functionality

What is software preservation?

Satisfactory preservation of software requires the consideration of the following four stages.

– **Storage**. A copy of software needs to be stored for long term preservation. Software is a complex digital object, with potentially a large number of components. There should be a strategy to ensure that the storage is secure and maintains coherence and authenticity, with appropriate strategies for storage replication, media refresh, format migration etc.
– **Retrieval**. A preserved software package to be retrieved at a date in the future, it needs to be clearly labelled and identified, with a suitable funcational catalogue so that the software can be retreived.
– **Reconstruction**. The preserved software should be reinstalled or rebuilt within a sufficiently close environment to the original so that it will execute. This is a complex operation, as there are a large number of contextual dependencies to the software execution environment which are required to be satisfied before the software will execute.
– **Replay**. In order to be useful at a later date, software needs be replayed, or executed and perform in a manner which is sufficiently close in its behaviour to the original. As with reconstruction, there may be environmental factors which may influence whether the software delivers a satisfactory level of performance.

In the first two aspects, software is much like any other digital object type. However, the problems of reconstruction and replay are key for software. Digital objects designed for human consumption have requirements for rendering which again have issues of satisfactory performance; science data objects also typically require information on formats and analysis tools to be "replayed" appropriately. However, software requires an additional notion of a environment with dependencies to hardware, other software, and build and configuration information.

Software Preservation Approaches

Various approaches to digital preservation have been proposed and implemented, usually as applied to data and documents. The Cedars Project (Cedars 2002) defined three main strategies, which we give here, and consider how they are applicable to software.

– **Technical Preservation. (techno-centric).** Maintaining the original software (typically a *binary*), and often hardware, of the original operating environment. Thus this is similar to the situation in museums where the original computing hardware is preserved and as much of the original environment is maintained as is possible. This approach is also taken in many legacy situations; otherwise obsolete hardware is maintained to keep vital software in operation.
– **Emulation (data-centric).** Re-creating the original operating environment by programming future platforms and operating systems to emulate that original environment, so that software can be preserved in binary and run "as is" in on a new platform.
– **Migration (process-centric).** Transferring digital information to a new platform. As applied to software, this means recompiling and reconfiguring the software source code to generate new binaries, apply to a new software environment, with updated operating system languages, libraries etc.

Software migration is a continuum. The minimal change is that the source code is recompiled and rebuilt directly from the original source. However in practice, the configuration scripts, or the code itself may require updating to accommodate differences in build systems, system libraries, or programming language (compiler) version. An extreme version of migration may involve rewriting the original code from the specification, possibly in a different language. However, there is not necessarily an exact correlation between the extent of the change and the accuracy of the preservation. Migration (or "porting" or "adaptive maintenance") is in practice how software which is supported over a long period of time is preserved. Long lasting software product teams spend much of their effort maintaining (or improving) the functionality of their system in the face of environment change.

These approaches have their advantages and disadvantages, which have been debated in the preservation literature. Technical (hardware) preservation has the minimal level of intervention and minimal deviation from the original properties of the software. However, in the long-term this approach is difficult to sustain as the expertise and spare components for the hardware become harder to obtain.

The emulation approach for preserving application software is widespread, and is particularly suited to those situations where the properties of the original software are required to be preserved as exactly as possible. For example, in document rendering where the exact pagination and fonts are required to reproduce the original appearance of the document; or in games software where the graphics, user controls and performance (e.g. it should not perform too quickly for a human player on more up to date hardware) are required to be replicated. Emulation is also an important approach when the source code is not available, either having been lost or not available through licensing or commercial restriction. However, a problem of emulation is that it transfers the problem to the (hopefully lesser) one of preserving the emulator. As the platform the emulator is designed for becomes obsolete, the emulator has to be rebuilt or emulated on another emulator. Nevertheless, emulation is being applied in several projects, notably within the European project PLANETS[4].

The migration approach does not seek to preserve all the properties of the original, or at least not exactly, only those up to the interface definition, which we could perhaps generalise to those properties which have been identified as being of significant for the preservation task in hand. Migration then can take the original source and adapt to the best performance and capabilities of the modern environment, while still preserving the significant functionality required. This is thus perhaps the most suited where the exact (in some respects) characteristics of the original are not required – there may be for example difference in user interaction or processing performance, or even software architecture – but core functionality is maintained. For example, for most scientific software the accurate processing of the original data is key but there is a tolerance to change of other characteristics.

The choice of preservation approach undertaken is thus dependent on the nature of the software artefacts available, the extent to which the original operating environment of the software can also be preserved or reproduced, and legal restrictions such as software licensing.

Performance Model and Adequacy

The test of the validity of a preservation approach is how a *performance* of the software adequately preserves some required characteristics. Performance as a model for the preservation of digital objects was defined in (Heslop et. al. 2002) to measure the effectiveness of a digital preservation strategy. Noting that for digital content, technology (e.g. media, hardware, software) has to be applied to data to render it intelligible to a user, they define a model where *Source data* has a *Process* applied to it, in the case of digital data some application of hardware and software, to generate a *Performance* for a user who extracts meaning from it. Different processes applied to a source may produce different performances but it is the properties of the performance which need to be considered for the value of a preservation action. Thus the properties can arise from a combination of the properties of the data with the technology applied in the processing. We consider how this model applies to software.

In the case of software, the performance is the execution of binary files on some hardware platform configured in some architecture to provide the end experience for the user. However, the processing stage depends on the nature of the software artefacts preserved which have differing reconstruction and replay requirements.

– If the binary is preserved, the process to generate the performance is one of preserving the original operating software environment and possibly the hardware too, or else emulating that software environment on a new platform. In this case, the emphasis is usually on performing as closely as possible to the original system.
– When source code and configuration scripts are preserved, then a rebuild process can be undertaken, using later compilers on a new platform, with new versions of libraries and operating system. In this case, we would expect that the performance would not necessarily preserve all the properties of the original (e.g. systems performance, or exact look and feel of the user interface), but have some deviations from the original.
– In an extreme case, a performance can be replicated by recoding the program in a different language. In this case, we would expect significant deviation from the original and perhaps only core functionality to be preserved

A software performance can thus result in some properties being preserved, and others deviating from the original or even being disregarded altogether. Thus in order to determine the value of a particular performance, in addition to the established notion of *Authenticity* of preservation (i.e. that the digital object can be identified and assured to be the object as originally archived) we define an additional notion of ***Adequacy***. *A software package (or indeed any digital object) can be said to perform adequately relative to a particular set of features ("significant properties"), if in a particular performance (that is after it has been subjected to a reconstruction and replay process) it preserves those significant properties to an acceptable tolerance.* By measuring the adequacy of the performance, we can thus determine how well the software has been preserved and replayed.

Performance of software and of data

A further feature of the performance model for software is that the measure of adequacy of the software is closely related to the performance of its input *data*. The purpose of software is to process data, so the *performance* of a software package becomes the *processing* of its input data. This relationship is illustrated in the modified performance model in *Figure 1*. There is also an interaction between the user and the software performance, reflecting the user's interaction with the software package during execution, changing the data processing and thus the data performance.

So for example, in the case of a word processing package which is preserved in a binary format, which itself is processed via operating system emulation, the performance of the package is the processing and rendering of word processing file format data into a performance which a (human) user can experience via reading it off a display. The user can then interact with the processing (via for example entering,

[4] http://www.planets-project.eu/

reformatting or deleting text) to change the data performance. Thus the measure of adequacy of the software is the measure of the adequacy of the performance when it is used to process input data, and thus how well it preserves the significant properties of its input data, and also perserving a known change in the data performance which results from user interaction with the processing.

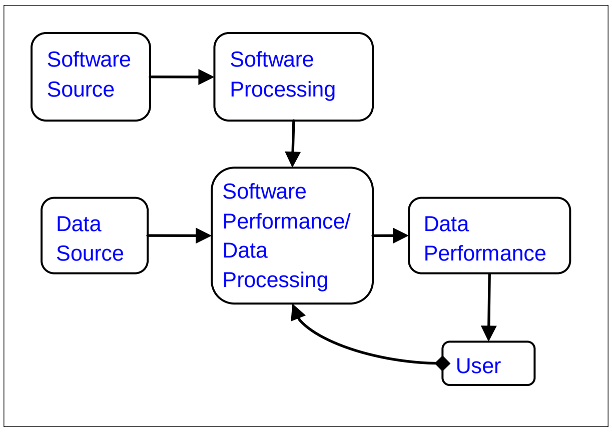

Figure 1: Performance model of software and its input data

Thus the adequacy of different preservation approaches is dependent upon the performance of the end result on the end use of *data*. As the software has to be able to produce an adequate performance for any valid input data, the adequacy can be established by performing trial executions against representative test data covering the range of required behaviour (including error conditions). The adequacy of preservation of a particular significant property can be established by testing against pre-specified suites of test cases with the expected behaviour, and pre-specified user interactions to change the data performance in known ways.

The follow table gives examples of properties and test cases to establish the adequacy of preservation appropriate for different categories of software, test after it has be reconstructed.

Software Category	**"Adequacy" Factor(s)**
Scientific Data Processing Software	The adequacy of the behaviour of this type of software may be measured by: – Running the software to process some pre-specified test input data – Comparing the output of the test run with the corresponding pre-specified test result; – Checking if the output exceeds the acceptable level of error tolerance for the software. For example, the NAG Software Library publishes test cases.
Games	The adequacy of the behaviour of a game may be measured by: – Comparing its User Interface UI with the screen capture of its original UI. – Comparing its performance against some pre-defined use cases. For example, the completion time of a particular level can be compared against the average completion time for that level in the original game. For example, when playing the emulated version of the 1990's DOS-based computer game Prince of Persia[5], some of the operations do not always work on the emulator and the original appearance of the game is also somewhat lost but it is still possible to play the complete game.
Programming Language Compilers	A compiler may be said to have been preserved adequately, if: – it covers all features of the programming language that it supports, e.g. concurrency (i.e. threads), polymorphism, etc. . – the application resulting from compiling its source code (written in a language supported by the compiler) using the compiler yields the expected behaviour. For example, some programming languages (e.g. Fortran, C, C++ etc.), have ISO standards[6] which describe the correct behaviour of a software written in these languages. These standards also provide test programs that may be used to assess the adequacy of a compiler for rendering all features of the programming language that it supports
Word Processor	The adequacy of a word processor may be measured based on its ability to: – render existing supported word documents with an acceptable level of error tolerance. For example, a word processor may be regarded as adequate as long as it clearly displays the contents (e.g. text, diagram, etc.) of a word document, even if some of the features of the document content, such as font colour and size, may have been rendered incorrectly or even lost completely.

[5] Best Old Games | Prince of Persia Download http://www.bestoldgames.net/eng/old-games/prince-of-persia.php

[6] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_tc_browse.htm?commid=45202

	– enable editing (e.g. add/change/remove text, change font) and saving existing word documents – enable creation and saving of new word documents For example, OpenOffice Word[7] is adequate for viewing and editing word documents originally created using Microsoft Word[8] with some level of error tolerance (e.g. images do not always appear as originally intended but viewable nevertheless)

A full conceptual model of software and detailing a set of properties for preservation which support all stages of the software preservation process (recall, reconstruction and replay has been developed within the project; which has been omitted for brevity. For details see (Matthews et.al. 2009a).

Applying the OAIS Reference Model to Software Preservation

The Reference Model for an Open Archival Information System (OAIS) is an ISO standard that is primarily concerned with the long-term preservation of digitally encoded information. In essence, the underlying notions of the OAIS reference model should be applicable to the long-term preservation of software artefacts as fundamentally (i.e. at bit level) they are in fact digitally encoded information. Therefore, as illustrated in Figure 2, the OAIS information model can be applied to the process of rendering a preserved Data Source on a future technological platform, where the rendering of the data requires the use of a particular software product, which in turn requires a specific complier, to be rebuilt from its preserved state. In short, the OAIS defined Descriptive Info, Representation Information (RI) and Preservation Description Information (PDI) (ISO 2002) can be used to retrieve (discover and access), reconstruct (compile source code), and replay (verify authenticity and run) a software object respectively.

However, once re-built, additional properties of the software are required to measure its adequacy for processing the Data Source, which in turn measures the performance of the compiler in re-building the software from its source code. Examples of these properties may include documentation of expected user interaction with the software in terms of expected inputs and outputs, information about accepted speed of execution and pre-defined test scripts and expected output and so on. This is not comprehensively addressed in the OAIS model but may be considered amongst the *Preservation Description Information* of software for demonstrating the satisfaction of significant properties, and thus viewed as an additional component of the OAIS information object in the context of long-term software preservation. Thus the notion of significant property corresponds to the proposed *Transformational Information Property* in (Giaretta et. al. 2009)

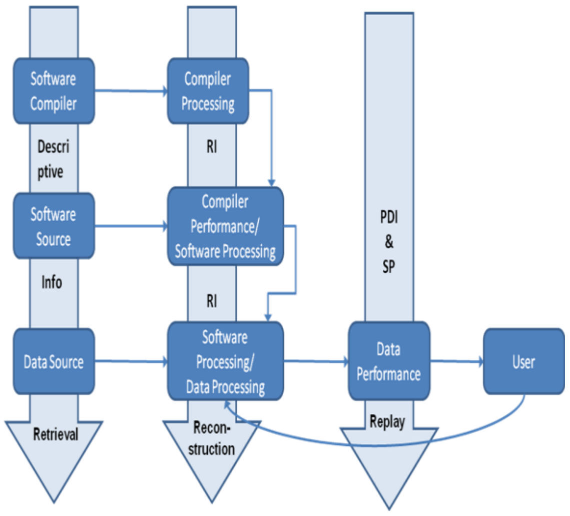

Figure 2: The OAIS Information Model and the Software Performance Model

Towards a methodology for software preservation

We have analysed the requirements for software preservation,considered the criteria which needs to be defined to ensure adequate preservation approach, and identified how software preservation can be related to the OAIS model. Consequently to perform a preservation analysis of a particular software product, we can use an OAIS compliant preservation analysis methodology, such as (Conway et. al. 2009). That paper presents an approach to preservation analysis which allows the software archiver to identify the designated community and appropriate information objects (including representation information, PDI and significant properties. Our framework identifies which components to look for in the case of software. Here we highlight some aspects particular to software which should inform the use of the methodology.

Stakeholder Analysis

As part of a principled preservation planning process, there is need to consider the stakeholders involved, with their different points of view and skills. The following table outlines some of the stakeholder categories which arise when considering software.

Stakeholder	Role.
Software creator	Has detailed knowledge of the software, so can provide reconstruction and replay properties, to make it easier to maintain software in the present as well as the longer term. This might be a single person, a team or a commercial company.
Software procurer	The funder of the software creator. The software may be required to work in a "live production" environment for analysing data for example, or may be a prototype only destined to prove the theorum of a research project. There

[7] http://www.openoffice.org/
[8]http://office.microsoft.com/en-gb/word/default.aspx

	are different expectations around the levels of documentation in these cases. For research projects, the procurer may be the funding council and rather remote; for live production software the procurer is likely to have a high degree of commitment to the project but is probably mostly interested in the outcomes of the software rather than the software per se.
Software user	Most users of software are not particularly interested in the software per se but the functionality that it provides. Whilst it provides the required functionality then it is needed to operate. This can cause issues when the software and the user are not changing requirements/functionality at the same rate.
Repository manager	In the research arena where all research outputs of an institution need to be collected and curated, then issues around software need to be considered, both for software in its own right and software as an access mechanism for data.

Risks analysis of preserving software

A key observation of our study is that preserving software is a complex process, with many dependencies and highly technical knowlede required to provide a preservable software product which can even be successfully reconstructed let alone adequately replayed. The most sucsseful software preservation activities were by those organisations who had initially developed software and then spent much time and effort to keep the software alive, using systematic software engineering methods, strong documentation, and a base of "tacit knowledge" in the workforce. In recognition that the best place to start preservation is during the creation of a digital object, we developed a tool which extended the popular software development tool Eclipse with the capability to record preservation properties during the development process, and in conjunction with the version control system of the software development.

However, it may not be possible to undertake the software preservation activity in conjunction with the development. Software preservation may be retrospective on legacy code, and the software product may be handed to an archivist or librarian to preserve. In this case, the risks associated with preservation are much greater, and a careful decision should be undertaken as to whether the additional effort is worthwhile. Figure 3 outlines some of the key descisions which must be undertaken to assess the value of a software preservation action. Again, the preservation properties identified in within our framework provide guideance to the required inforamtion and therefore allow the value of the preservation action to be assessed.

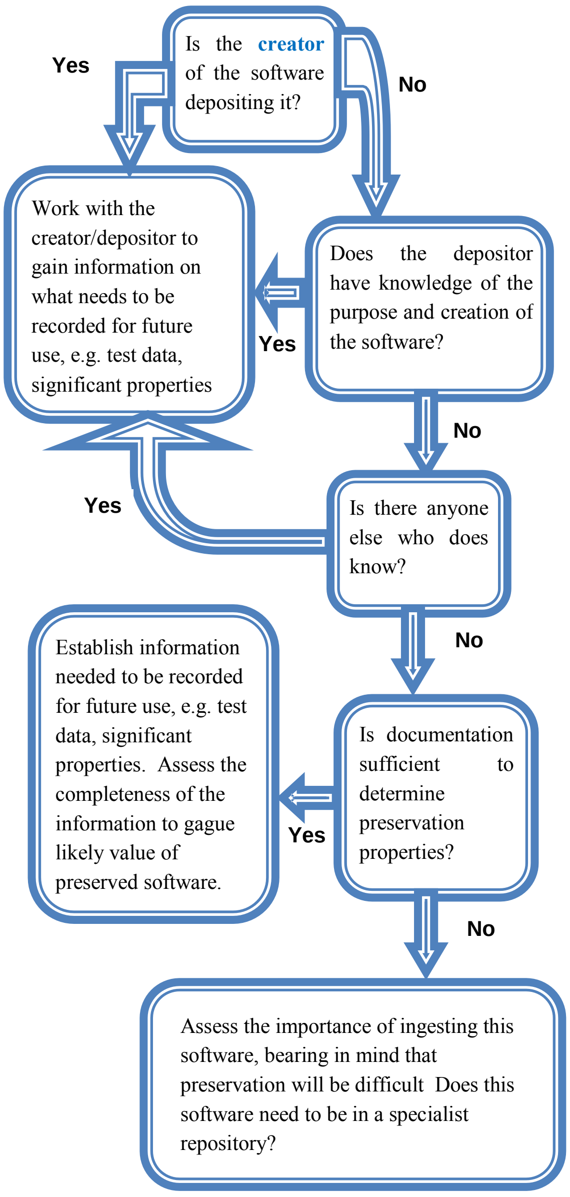
Is the creator of the software depositing it?
Yes
No
Work with the creator/depositor to gain information on what needs to be recorded for future use, e.g. test data, significant properties
Does the depositor have knowledge of the purpose and creation of the software?
Yes
No
Yes
Is there anyone else who does know?
No
Establish information needed to be recorded for future use, e.g. test data, significant properties. Assess the completeness of the information to gague likely value of preserved software.
Yes
Is documentation sufficient to determine preservation properties?
No
Assess the importance of ingesting this software, bearing in mind that preservation will be difficult Does this software need to be in a specialist repository?

Figure 3: Descision proceedure for software preservation analysis.

Validating the Approach

In order to validate our framework and methodology for software preservation, we have undertaken trials in the application of the software preservation framework in the context of a use case involving the British Atmospheric Data Centre[9]. This includes evaluating the overall efficiency of the framework against a number of BADC software, specifically in terms of its relevance (to the software that it is applied to) and sufficiency (of the information recorded) for long-term preservation of software, considered within the context of the BADC's approach to accommodating changes in the technological environment to ensure effective long-term software maintenance and re-use.

In short, the BADC study highlights that the BADC should benefit from suitable software preservation models and tools that would effectively manage

[9] http://badc.nerc.ac.uk/home/index.html

complexity and reduce costs of software preservation, and could also be integrated within existing systems for software development and maintenance.

The preservation approach should be assisted with tool support to allow the systematic collection of preservation propertied for software. We have developed a Java-based tool called Significant Properties Editing and Querying for Software (SPEQS), developed in view of our analysis of the BADC use case. SPEQS demonstrates the feasibility of integrating preservation (in terms of capturing preservation properties identified in the software preservation framework) within the software development lifecycle to aid its long-term preservation in future, and thus has the potential to be beneficial to software development oriented organisations. The BADC case study and the SPEQS tool are described in (Matthews et. al. 2009b)

A further validation of the approach took place within a preservation exercise for solar-terrestrial physics data. This study considered raw data which could be analysed to extract an Ionogram – a graph showing ionization layers in the atmosphere. Current scientists use a software product called SAO explorer[10] to extract ionograms from the data. This software was archived in accordance with the methodology described in this paper. As a result the archived software could with confidence be integrated into an larger OAIS compliant solution for the preservation of mmm data files. This solutions permits the long term study of specified atmospheric phenomena from this geographic location. The archived SAO explorer solution could also then be deposited in the DCC registry repository of representation information thereby providing a solution which can be re-used by hundreds of ionosphere monitoring station which are active globally.

Conclusions

We can see that there are good reasons to consider the preservation of software. Further, we have established a reference framework to discuss software preservation in terms of an abstract performance model. Software is not an independent entity in its own right, but only can be judged to be adequately preserved in the context of the satisfactory preservation of its target data objects. Different preservation approaches can be adopted which can execute binaries directly, can emulate the software, or carry out software migration by recompiling source code, or even recoding. All can in different circumstances support good preservation, and the proposed performance model can be used to judge the adequacy of the preservation approach. This adequacy test is akin to the software testing process familiar to software engineers. Indeed, good software engineering practice is likely to be a fruitful source of techniques for good software preservation.

Software engineering best practice shares many of the concerns of software preservation in order to produce quality software which can be maintained and reused in the future, such providing version control, dependency analysis and good documentation. We consider how software preservation can be integrated into the software lifecycle to systematically capture those properties required for preservation and an adequate replay of the software.

[10] http://ulcar.uml.edu/SAO-X/SAO-X.html

Acknowledgements

This work was carried out under the UK JISC study into the Significant Properties of Software and also the JISC sponsored Tools and Guidelines for Preserving and Accessing Software Research Outputs. The SAO Explorer case study was undertaken in conjunction with the DCC Scarp project. We would like to thank our colleagues David Giaretta, Steven Rankin and other members of the Digital Curation Centre and CASPAR projects for their advice and discussions.

References

The Cedars Project. 2002. *The Cedars Guide to Digital Preservation Strategies*. Retrieved July 29, 2008, from http://www.leeds.ac.uk/cedars/guideto/dpstrategies/dpstrategies.html

Conway, E,; Giaretta, D.; and Dunckley, M.. 2009. Curating scientific research data for the long term: a preservation analysis method in context. In *proceedings of iPres 2009, The 6th International Conference on Preservation of Digital Objects*

Giaretta, D.; Matthews, B.; Bicarregui, J.; Lambert, S.; Guercio, M,; Michetti, G.; and Sawyer, D. 2009. Significant Properties, Authenticity, Provenance, Representation Information and OAIS. In *proceedings of iPres 2009, The 6th International Conference on Preservation of Digital Objects*

Heslop, H.; Davis, S.; and Wilson, A. 2002. *An Approach to the Preservation of Digital Records*, National Archives of Australia, 2002. Retrieved July 29, 2008, from http://www.naa.gov.au/Images/An-approach-Green-Paper_tcm2-888.pdf

ISO 2002. Reference Model for an Open Archival Information System (OAIS). *Recommendation for Space Data Systems Standard, CCSDS Blue Book*. http://public.ccsds.org/publications/archive/650x0b1.pdf

Matthews, B.M.; Bicarregui J.C.;. Shaon, A.; and Jones, C.M. 2009a. A Framework for the Significant Properties of Software . *JISC Tools and Methods for Software Preservation Project report* http://epubs.stfc.ac.uk/work-details?w=51076

Matthews, B.M.; Shaon, A.; Bicarregui J.C.;. Jones, C.M.; and Woodcock, J. 2009b. An Approach to Software Preservation. In *proceedings PV 2009 Ensuring Long-Term Preservation and Adding Value to Scientific and Technical Data,* to appear.

Zabolitzky, J.G. 2002. Preserving Software: Why and How. *Iterations: An Interdisciplinary Journal of Software History, 1*. Retrieved July 29, 2008, from http://www.cbi.umn.edu/iterations/zabolitzky.html