An Approach to Software Preservation

Brian Matthews[1], Arif Shaon[1], Juan Bicarregui[1], Catherine Jones [1] , Jim Woodcock [2]

[1] e-Science Centre, Science and Technology Facilities Council

STFC Rutherford Appleton Laboratory, Chilton, Didcot, OXON OX11 0QX

EMail: brian.matthews@stfc.ac.uk

[2] Department of Computer Science, University of York
Heslington, York, YO10 5DD, UK

ABSTRACT

Long-term preservation of software components is a key aspect of preservation of data, as software required for processing and analyzing data also needs to be preserved in order to maintain the re-usability of data in future. However, software preservation to date is a relatively underexplored topic of research and frequently seen as a secondary activity, mainly due to the inherent complexity of software artifacts, which is generally deemed a major barrier to their preservation. In this paper, we present a conceptual framework to capture and organise the main notions of software preservation, which are required for a coherent and comprehensive approach. In particular, the framework introduces a notion of *adequacy* of preservation, an aspect of the OAIS concept of authenticity which tests the future performance of software against specified preservation properties. We also evaluate the application of the software preservation framework in the context of a use case involving the British Atmospheric Data Centre (BADC).

Keywords: Software Preservation, Software Performance, Adequacy, Authenticity, BADC

INTRODUCTION

Preservation of software components is a key aspect of preservation of data, as processing and analysis software frequently needs to be preserved to maintain the usability of data. However, 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. Further, the preservation of software is frequently seen as a secondary activity and one with limited usefulness.

In this paper, we discuss some of the motivations and approaches taken to preserve software. We also discuss some framework concepts of what it means to preserve software, in particular a notion of *adequacy* of preservation, an aspect of the OAIS concept of authenticity which tests the future performance of software against specified preservation properties. In addition, we analyse the relationship of this software preservation framework with the different components of the OAIS information model in terms of their applicability to the retrieval, reconstruction and replay of software on a future technological platform. We then identify within the framework, a number of additional properties of software against which the adequacy of its behaviour, and hence its preservation may be measured in future.

We go on to discuss the application of the software preservation framework in the context of a use case involving the British Atmospheric Data Centre. 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.

WHY PRESERVE SOFTWARE?

A key question to answer with respect to preservation of software is why it is a useful thing to do. After all, software is known to be both very fragile and very disposable, especially when changes occur in related environment, such as hardware, operating system, versions of systems (e.g. programming languages and compilers) and configuration change. For example, compiling with a different floating point module may produce quite different results in the analysis. Also, in the face of environment change and the complexity of large-scale systems, developers often throw away previous software and start again from scratch, as it may be easier to write new software (given the original *data* is preserved) rather than wrestle with legacy.

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.

Preserve a complete record of work: Software is frequently an output of research, where it is typically deemed as means of testing the hypothesis of research. This is particularly the case in Computer Science. In such cases, software should be preserved along with other research outputs, e.g. thesis to ensure preservation of the complete work.

Preserving the data: Software may need to be preserved to support the preservation of data and documents, to keep them live and reusable. For example, the data holding for NASA's Planetary Data System (PDS) Geosciences Node relies on various web-based services and software for its effective discovery and accurate use. Development of these services is time-consuming and requires intimate knowledge of the mission and the data, which may be impossible or extremely difficult and expensive to reproduce in future. Therefore, ensuring long-term reusability of this data holding would need these services and software to be properly preserved [1].

Handling Legacy: Perhaps the prime motivation to preserve software for most organisations is to save effort in recoding. Legacy code still needs to be used, due to its specialised function or configuration and 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 a significant part of the effort which is undertaken by software developers both in-house within end-user organisations, and also within software houses.

A CONCEPTUAL FRAMEWORK FOR LONG-TERM SOFTWARE PRESERVATION

We have developed a conceptual framework that is intended to comprehensively address the core aspects of software preservation and identify the software properties needed to retrieve a software artifact in future, reconstruct it and measure the adequacy of its preservation for the purpose of replaying (i.e. reusing) it on a future technological platform. The framework comprises the following two conceptual models that express these notions:

A Conceptual Model for Software

We recognize that a conceptual data model is required to determine the level of granularity at which preservation properties of software can be identified, and provide an understanding of the relationship between digital objects, thus giving traction on handling the complexity of the objects, a particularly important aspect in handling software. Therefore, we have developed a general model for software digital objects that is intended to provide a comprehensive view of the underlying dependencies of software, and thus help identify its preservation properties. The model consists of four major conceptual entities which together describe a complete ***Software System***. These are *Product, Version, Variant* and *Instance* (Figure 1).

Product: The product is the whole top-level view of the system, and is how the system may be commonly or informally referred to. Products can vary in size and can range from a single library function (e.g. a function in the NAG library [2]), to a very large system with multiple sub-products with independent provenances (e.g. Linux).

Version: A version of a software product is an expression of the product which provides a single coherent presentation of the product with a well defined functionality and behaviour. Note also that in composite products, the sub-products will themselves have a number of versions which will be related to versions of the complete product. These releases will not necessarily be synchronised, so the relationship between versions of sub-products need to be captured.

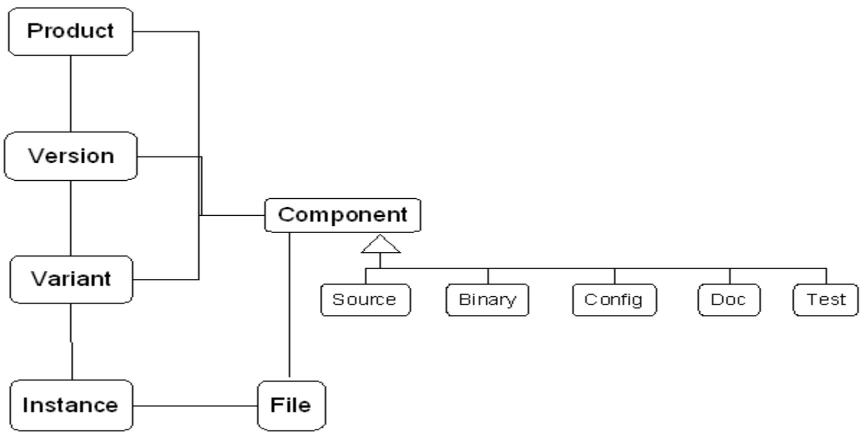

Figure 1: The Software Component Conceptual Model

Variant: Versions may have a number of different variations to accommodate different operating environments, thus we define a *Variant* of the product to be a manifestation of the system which changes in the *software operating environment*, for example target hardware platform, target operating system, library, and programming language version. In this case, the functionality of the version is maintained as much as is practical; however, due to different behaviour supported by different platforms, there may be variations in behaviour, in error conditions and user interaction (e.g. the look and feel of a graphical user interface). It may be arguable that in some circumstances Versions are subordinate to Variants, and in others we may wish to omit one of these stages such as software which is only ever targeted at one platform. But it is worth distinguishing the two levels, as it makes a distinction between adaptations of the system largely to accommodate change in *functional properties (versions),* with those which are largely to accommodate change in *properties of the operating environment (variants).*

Instance: An actual physical instance of a software product which is to be found on a particular machine is known as an *Instance.* It may be also referred to as an installation, although there is no necessity for the product to be installed; a master copy of stored at a repository under a source-code management system may well not be executable within its own environment.

All of the entities in the above conceptual model of software which form a software system are *composite*. Some of them may be subsystems, with sub-products. All systems however, will be constructed out of many individual *components*, e.g. source, binary, etc. (Figure 1). A component is a storable unit of software which when aggregated and processed appropriately, forms the software system as a whole. Logical components typically (but not necessarily always) roughly corresponds with a physical *file* (a unit of storage within an operating system's memory management). However, multiple components can be stored within in one file (e.g. a number of subroutines within one file) or across a number of files (e.g. help system or tutorial stored within a number of HTML files). Components may also be formed of a number of different digital objects, (e.g. text files, diagrams, sample data) which themselves would have preservation properties associated with their data format. An effective preservation strategy for the full software system would have to consider those digital objects as well.

Software Performance Model

Given the uncertainty of long-term digital preservation, it is necessary to be able to measure the effectiveness of a digital preservation strategy. In the case of software we propose to base this on the notion of how a sufficient level of *performance* preserves the required characteristics of software. For example, in the case where a software binary is preserved, the process generating its performance requires 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. On the other hand, when source code and configuration and build scripts of a software product are preserved, then a rebuild process can be undertaken, using later compilers and linkers on a new platform, with new versions of libraries and operating systems. 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. Thus, a software performance can result in some properties being preserved and others deviating from the original or even being disregarded altogether. Therefore, in order to determine the value of a particular performance, we define a notion of ***Adequacy*** for software, *which can be said to perform* ***adequately*** *relative to a particular set of features perceivable by the user (or another software agent) ("significant properties"), if in a particular performance (that is after it has been subjected to a particular process) it preserves that set of significant properties to an acceptable tolerance.*

This notion of adequacy is usually viewed as an aspect of the established notion of ***Authenticity*** of preservation, which signifies that the digital object can be identified and assured to be the object as originally archived. However, authenticity of preservation does not also guarantee a reliable behaviour from the software once reconstructed in future; it might incur a loss of some of its original features during its reconstruction process. However, the software could still be used for the remaining features retained after reconstruction, which could be sufficient to extract an acceptable level of performance from the software. An example of such software is the emulated version of the 1990's DOS-based computer game Prince of Persia [3]. The term *Adequacy* introduced here is intended to represent this particular concept.

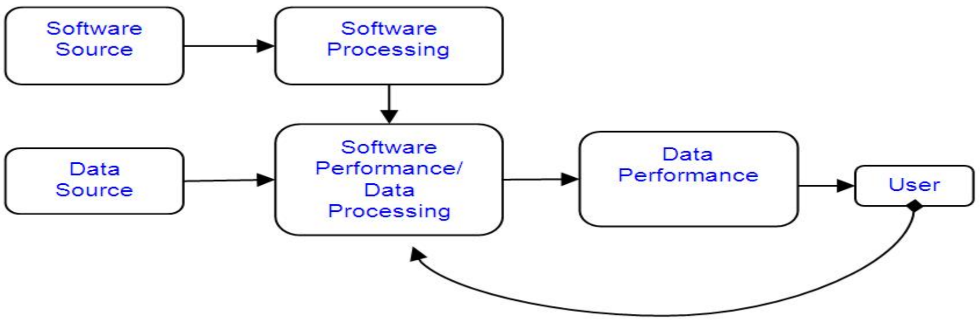

Figure 2: Performance model of software and its input data

We express this notion of adequacy of software performance in the form of a conceptual model, which is based on a performance model for the preservation of digital objects, defined by the National Archives of Australia to measure the effectiveness of a digital preservation strategy [4]. In general, the model illustrates the relationship between software and its input data as in Figure 2, where the reversed the arrow between software performance and user reflects the user's interaction with the software product during execution, changing the data processing and thus its performance. So for example, in the case of a word processing product which is preserved in a binary format that is processed via operating system emulation, the performance of the product 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, reformatting or deleting text) to change the data performance. Thus the measure of adequacy of the software is the satisfaction of the performance to the user when it is used to process input data, and thus how well it preserves the significant properties of its input data, and also preserving a known change in the performance which results from user interaction with the processing.

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 3, 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) [5] can be used to retrieve (discover and access), reconstruct (compile source code), and replay (verify authenticity and run) a software object respectively.

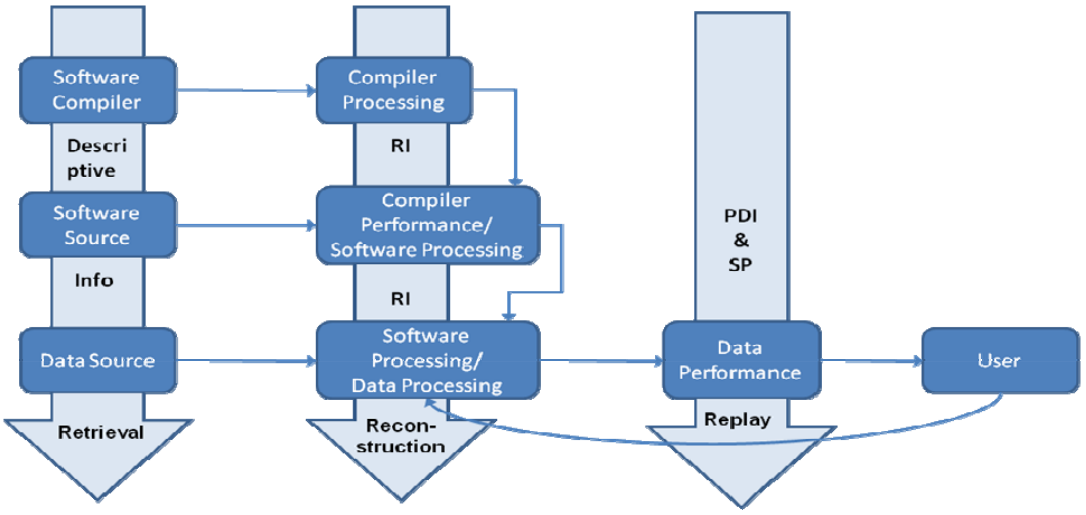

Figure 3: The Relationship between the OAIS Information Model and the Software Performance Model

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.

The preservation framework attempts to identity these additional properties of software along with other OAIS equivalent preservation properties of software for each of the four major conceptual entities of software defined in the conceptual model for software discussed earlier in the paper. Details of these preservation properties of software can be found in [6].

THE BADC CASE STUDY

The National Centre for Atmospheric Science's British Atmospheric Data Centre (BADC) [7] is a NERC Designated Data Centre which currently has over 250TB of atmospheric data for the consumption of UK scientists and researchers [8]. In order to facilitate efficient accessibility and usability of these large volumes of atmospheric data, the BADC also develops, supports, and provides access to a variety of software, which ranges from very simple data conversion tools to highly complex weather prediction models. Considering the importance of the BADC software in enabling accessibility and interpretation of its large data holdings, effective long-term preservation (i.e. re-use) of the BADC datasets implies the need for appropriate preservation actions for its software. We consulted with the

BADC software development and maintenance team to analyse their approaches to software maintenance in the context of a number of their software products, such as the BADC Web Feature Service (WFS)[1]. This indicated that long-term preservation of these software products is not considered within the current operational remit of the BADC. On the whole, the BADC considers the long term archiving of software an impractical option principally due to the complex dependencies of software. It takes the view that it expects much current software will be superseded by newer software which will be capable of recreating and enhancing much of the existing analysis and access functionality. Further, the BADC considers the costs of preserving software, especially in terms of migrating to newer technological platform, to be a prohibitive factor and, hence outside their current remit.

In general, the BADC case study reinforces the view that the inherent complexity of software is a significant barrier to its long-term preservation. In addition, it highlights another prohibitive factor for long-term preservation of software: *cost of preservation*. Effective preservation of a digital object over the long-term requires its continuous management and enhancement over its lifecycle. This involves, amongst other tasks, periodically assessing (and improving) the adequacy of the preservation strategy for ensuring effective re-construction and re-use of the digital object notwithstanding any related technological changes. This is expected to impose significant recurring costs on the organisation undertaking long-term preservation, in terms of technical resources, personnel effort etc. required. There likely to be even greater effort and hence costs required for the long-term preservation of software due to the complex dependencies between its components. Thus, for an organisation, such as the BADC who is already bearing the costs of maintaining and developing a wide array of software, it would be difficult to justify and incorporate within its current remit and budget, the additional costs of long-term software preservation as such an activity might not be deemed beneficial to BADC in the short-term. In addition, the high complexity and costs of employing currently available preservation mechanisms, such as migration and emulation would also add to the overall costs of long-term preservation of software.

Therefore, we envision that the organisations, such as the BADC should benefit from the conceptual framework for software preservation presented in this paper. The framework provides a comprehensive and organised view of the underlying dependencies of software in the context of preservation and facilitates accurate identification of the software properties needed for its effective preservation. This can aid in efficient management of the complexity of software preservation, which in turn could help reduce the overall costs of preservation. Additionally, the framework could potentially be used for incorporating long-term preservation functions into existing systems for software development and maintenance, such as the one at BADC.

Evaluating the Software Preservation Framework against BADC Software

We have evaluated the framework against a number of BADC software, such as the BADC Web Feature Service (WFS). For this, we tried to collect the appropriate value(s) for each of the preservation properties defined in the framework for each major conceptual entity of the BADC WFS. [9] details the results of this exercise.

The experience of applying the framework for software preservation to the BADC software has shown that the framework is sufficiently relevant to the software used as well as being adequate in terms of the information recorded. However, it has also highlighted the necessity to have considerable knowledge of both the framework and software in question to accurately apply the framework to the software. This indicates a need for tools to facilitate the recording of software preservation properties by providing guidelines which, for example, explain the underlying concepts of the framework in a user-friendly manner.

[1] The BADC WFS Enables retrieving and updating geospatial data encoded in Geographic Markup Language (GML - http://www.opengeospatial.org/standards/gml), or any GML-based formats, irrespective of the location or storage media of the data. The implementation is based on the Open Geospatial Community (OGC) standard for Web Feature Service(http://www.opengeospatial.org/standards/wfs)

We have attempted to address the aforementioned issue with the software preservation framework by developing a tool, namely **Significant Properties Editing and Querying for Software** (SPEQS). In effect, SPEQS aims to demonstrate the feasibility of incorporating capturing preservation properties of software within the software development lifecycle to aid its long-term preservation. It has been implemented in Java as a plug-in for Eclipse, a widely used Open Source interactive software development environment, to enable software developers to record, edit and query preservation properties of software directly from within the Eclipse environment. This approach of enabling the developer(s) of a software project to record its preservation properties is envisaged to contribute towards ensuring the accuracy of the information recorded.

CONCLUSION

In this paper we have presented a conceptual framework to express a rigorous approach to long-term software preservation. We believe that this is a general and principled approach which can cover the preservation needs of a wide range of different software products (e.g. the BADC software products), including modern distributed systems and service oriented architectures, which are typically built of pre-existing frameworks and have a large number of dependencies on a widely distributed network of services, many of which are outside the control of the typical user (e.g. DNS services, proxies). We also believe that the performance model presented here, which introduces a notion of ***Adequacy*** of software performance as well as a notion of user feedback to influence the performance represents an approach to preserving the user interface and the user interaction model, although work is required to further develop that notion. In addition, further work is required to evaluate the preservation framework, especially against a range of software types to cover the diversity of software and to consider how to support the preservation of legacy software.

REFERENCES

[1] - T. Stein, E. Guinness, S. Slavney: *Methods for Archiving and Distributing Science Data from Planetary Missions*. PV2007 Conference, DLR (2007). http://www.pv2007.dlr.de/Papers/Stein_methods_for_archiving.pdf

[2] - Numerical Algorithm Group http://www.nag.co.uk/

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

[4] - Heslop, H., Davis, S., Wilson, A. (2002). *An Approach to the Preservation of Digital Records*, National Archives of Australia, 2002. http://www.naa.gov.au/Images/An-approach-Green-Paper_tcm2-888.pdf

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

[6] - A Framework for the Significant Properties of Software - http://sigsoft.dcc.rl.ac.uk/twiki/bin/view/Main/SigSoftFramework

[7] - British Atmospheric Data Centre - http://badc.nerc.ac.uk/home/index.html

[8] - A. Stephens, A. Rudder, A. Iwi, K. Marsh and J. Kettleborough, *BADC services and datasets for climate models*, , Poster - http://www.noc.soton.ac.uk/coapec/pdfs/FM/P15Marsh.pdf

[9] - Applying the SP Framework to BADC WFS/GeoServer, twiki page. http://sigsoft.dcc.rl.ac.uk/twiki/bin/view/Main/SigSoftExamples#Applying_the_SP_framework_to_the

ACKNOWLEDGEMENT

We would like to thank our colleagues David Giaretta, Esther Conway, Steven Rankin, Brian McIlwrath and other members of the Digital Curation Centre and CASPAR projects for their advice and discussions, and the BADC for their assistance with the case study. The work was carried out under the Joint Information Systems Committee (JISC) study into the Significant Properties of Software and the JISC project Tools and Guidelines for Preserving and Accessing Software Research Outputs.