Software Preservation

Benefits framework

CC443D006-1.0
7 December 2010
Cover + 74 pages

Neil Chue Hong
Steve Crouch
Simon Hettrick
Tim Parkinson
Matt Shreeve

Software
Sustainability
Institute

curtis+cartwright

Software Sustainability Institute
The University of Edinburgh
James Clerk Maxwell Building
Mayfield Road
Edinburgh EH9 3JZ

tel: +44 (0) 131 650 5030
email: info@software.ac.uk
web: http://www.software.ac.uk

Curtis+Cartwright Consulting Ltd
Main Office: Surrey Technology Centre,
Surrey Research Park, Guildford
Surrey GU2 7YG

tel: +44 (0)1483 685020
fax: +44 (0)1483 685021
email: postmaster@curtiscartwright.co.uk
web: http://www.curtiscartwright.co.uk

Registered in England: number 3707458

Registered address:
Baker Tilly, The Clock House,
140 London Road, Guildford,
Surrey GU1 1UW

Summary of framework

1 An investigation of software preservation has been carried out by Curtis+Cartwright Consulting Limited, in partnership with the Software Sustainability Institute (SSI), on behalf of the JISC.[1] The aim of the study was to raise awareness and build capacity throughout the Further and Higher Education (FE/HE) sector to engage with preservation issues as part of the process of software development. Part of this involved examining the purpose and benefits of employing preservation measures in relation to software, both at the development stage and retrospectively to legacy software. The study built on the JISC-funded 'Significant Properties of Software' study[2] that produced an excellent introduction and comprehensive framework to software preservation.

2 This is a framework document that assists developer groups and their sponsoring bodies to understand and gauge the benefits or disbenefits of allocating effort to:

- ensuring that preservation measures are built into software development processes;
- actively preserving legacy software.

3 We have condensed the key information from the framework into a two-side crib sheet[3]; this document is the full, detailed version intended for reference.

Purposes, benefits and scenarios

4 A key challenge in digital preservation is being able to articulate, and ideally prove, the need for preservation. A clear framework of purposes and benefits facilitates making the case for preservation. Our framework also includes a range of scenarios for each purpose to give some illustrative examples of where the purpose and accompanying benefits might be relevant.

Purpose	Benefits	Scenarios
Encourage software reuse	Reduced development cost Reduced development risk Accelerated development Increased quality and dependability Focused use of specialists Standards compliance Reduced duplication Learning from others Opportunities for commercialisation	Continuing operational use in institution Increasing uptake elsewhere Promoting good software

1 <http://www.jisc.ac.uk/fundingopportunities/funding_calls/2010/02/softwarepreservation.aspx> accessed 4 April 2010.

2 *The Significant Properties of Software: A Study*, Matthews et al, STFC, March 2008.

3 See <http://www.software.ac.uk/resources> for the crib sheet and related materials.

Purpose	Benefits	Scenarios
Achieve legal compliance and accountability	Reduced exposure to legal risks Avoidance of liability actions Easily demonstrable compliance lessens audit burden Improved institutional governance Enhanced reputation	Maintaining records or audit trail Demonstrating integrity and authenticity of data and systems Addressing specific contractual requirements Addressing specific regulatory requirements Resolving copyright or patent disputes Addressing the need to revert back to earlier versions due to IP settlements Publishing research openly for transparency Publishing research openly as a condition of funding
Create heritage value	(Heritage value is generally considered to be of intrinsic value)	Ensuring a complete record of research outputs where software is an intermediate or final output Preserving computing capabilities (software with or without hardware) that is considered to have intrinsic value Supporting the work of museums and archives
Enable continued access to data and services	For research data and business intelligence: – Fewer unintentional errors due to increased scrutiny – Reduced deliberate research fraud – New insight and knowledge – Increased assurance in results For systems and services: – Current operations maintained – Opportunity for improved operations via corrective maintenance – Reduced vendor lock-in – Improved disaster recovery response – Increased organisational resilience – Increased reliability	Reproducing and verifying research results Repeating and verifying research results (using the same or similar setup) Reanalysing data in the light of new theories Reusing data in combination with future data 'Squeezing' additional value from data Verifying data integrity Identifying new use cases from new questions Maintaining legacy systems (including hardware) Ensuring business continuity Avoiding software obsolescence Supporting forensics analysis (*eg* for security or data protection purposes) Tracking down errors in results arising from flawed analysis

5 We recommend that these purposes and benefits be combined with preservation plans regarding data and hardware: digital preservation should be considered in an integrated manner. For example, media obsolescence and recovery is often as much a part of a software preservation project as a data preservation project.

Does this software need preserving?

6 We do not believe that there is a simple and universally applicable formula for determining if your software needs to be preserved, and how to go about preserving it, so instead we present thought-provoking questions and a range of factors which should be taken into account. This should be read through and careful consideration given to those aspects relevant to the software you are interested in.

7 The following questions should be considered:

- Is the software covered by a preservation policy / strategy?
- Is there a clear purpose in preserving the software?
- Is there a clear time period for preservation?
- Do the predicted benefit(s) exceed the predicted cost(s)?
- Is there motivation for preserving the software?
- Is the necessary capability available?
- Is the necessary capacity available?

8 Note that if at all possible, especially where the software is an enabler, it's advisable to turn a software preservation problem into a data preservation problem. These problems are invariably easier to handle.

How should your software be preserved?

9 Seven different options for preservation and sustainability are presented:

- **Technical preservation (techno-centric)** - Preserve original hardware and software in same state;
- **Emulation (data-centric)** - Emulate original hardware / operating environment, keeping software in same state;
- **Migration (functionality-centric)** - Update software as required to maintain same functionality, porting/transferring before platform obsolescence;
- **Cultivation (process-centric)** - Keep software 'alive' by moving to a more open development model, bringing on board additional contributors and spreading knowledge of process;
- **Hibernation (knowledge-centric)** - Preserve the knowledge of how to resuscitate/recreate the exact functionality of the software at a later date;
- **Deprecation** - Formally retire the software without leaving the option of resuscitation/recreation;
- **Procrastination** - Do nothing.

10 The following questions should be considered:

- How much access do you have? (Owner / developer / access to source code / access to hardware / user)
- Do you have the necessary Intellectual Property Rights (IPR)?

- What are you needing to preserve? (A few major pieces of functionality / Most of the functionality, but tolerant of minor deviations / All functionality, but fixing errors when found / Must perform exactly as original)
- What is your likely effort profile? (Something/nothing now, something/nothing in the future)
- What is the maintainability of underlying hardware?
- Is maintaining integrity and/or authenticity an important requirement?
- How long do you want to preserve it for?
- Can you afford it?
- Are you also interested in further development or maintenance?
- What development effort has been invested into the software so far?
- Is the software already open source, or could it be made open source?
- Are there any barriers to making it open source?
- Is the proposed approach appropriate to every purpose?
- What are the relative advantages and disadvantages of each approach under consideration?

Document history

Version	Date	Description of Revision
0.1	4 October 2010	Incomplete draft circulated for internal progress review
0.2	8 October 2010	Draft for internal review and contributions by the study team
0.3	14 October 2010	Draft for independent review
0.4	15 October 2010	Draft for client review
1.0	7 December 2010	Issue release

This page is intentionally blank

List of contents

This page is intentionally blank

List of abbreviations

API	Application Programming Interface
CAD	Computer Aided Design
CARET	Centre for Applied Research in Educational Technologies
CCG	Computational Chemistry Group
CCPs	Collaborative Computational Projects
CIO	Chief Information Officer
COG	Component Obsolescence Group
COTS	Commercial Off The Shelf
CPOSS	Crystal Prediction of the Organic Solid State
CSIRO	Commonwealth Scientific and Industrial Research Organisation
CTO	Chief Technology Officer
DCC	Digital Curation Centre
DExT	Data Exchange Tools and Conversion Utilities
DPHEP	Data Preservation in High Energy Physics
EGRET	Engaging Responses to Emerging Technologies
EPSRC	Engineering and Physical Sciences Research Council
FAA	Federal Aviation Administration
FE	Further Education
GISS	Goddard Institute for Space Studies
HE	Higher Education
HEP	High Energy Physics
HPC	High-Performance Computer
IO	Input / Output
IPAC	Infrared Processing and Analysis Center
IPR	Intellectual Property Rights
IR	Infrared
JAR	Joint Aviation Requirements
LOTAR	LOng Term Archiving
LTA	Long-Term Archiving
MOCA	Mitigation of Obsolescence Cost Analysis
OA	Open Access
OAIS	Open Archival Information Systems
OEM	Original Equipment Manufacturer
OSS	Open Source Software
OU	Open University
PARSE	Permanent Access to the Records of Science in Europe
PDM	Product Data Management

PMC	Project Management Committee
RESL	Re-usable Educational Software Library
RRL	Reuse Readiness Levels
SaaS	Software-as-a-Service
SoURCE	Software Use, Re-use & Customisation in Education
SPEQS	Significant Properties Editing and Querying for Software
SSI	Software Sustainability Institute
STFC	Science and Technology Facilities Council
UCL	University College London
UKDA	UK Data Archive

1 Introduction

1.1 General

1.1.1 An investigation of software preservation has been carried out by Curtis+Cartwright Consulting Limited, in partnership with the Software Sustainability Institute (SSI), on behalf of the JISC.[4] The aim of the study was to raise awareness and build capacity throughout the HE/FE sector to engage with preservation issues as part of the process of software development. Part of this involved examining the purpose and benefits of employing preservation measures in relation to software, both at the development stage and retrospectively to legacy software. This study was undertaken between April 2010 and October 2010.

1.1.2 This framework document forms one of a series of outputs from the project.[5] This version of the document (V1.0) is for public release. The content in this document is licensed under an Attribution-ShareAlike 2.0 UK: England & Wales[6]. The rights to the design, layout and logos in this report are wholly retained by the authors.

1.1.3 This framework document is intended for the varied and numerous groups working within (or in collaboration with) the UK HE/FE community with a non-exclusive but primary focus towards those working with open-source software.

1.2 Acknowledgements

1.2.1 The project team would like to thank everyone who contributed to this study. In particular the SigSoft project team[7] at the Science and Technology Facilities Council (STFC); Ross Gardler at OSS Watch[8]; and the Component Obsolescence Group (COG)[9] and Graeme Rumney (Sellafield Limited) for their prior and parallel work in the area. Their work has provided a solid basis for the main thrust of this study – namely to raise awareness. We see no need to duplicate their excellent materials and would recommend them to all those interested in the topic.

1.3 Objectives

1.3.1 This is a framework document that assists developer groups and their sponsoring bodies to understand and gauge the benefits or disbenefits of allocating effort to:

- ensuring that preservation measures are built into software development processes;
- actively preserving legacy software.

1.3.2 The intention is that deeper understanding enables the reader to make better decisions about the practicalities of software preservation. Because understanding the benefits by themselves does not lead to better decisions, and because the exact mix of benefits depends on the particular set of activities proposed rather than the end outcome, this framework also covers costs and approaches to software preservation.

---

4 <http://www.jisc.ac.uk/fundingopportunities/funding_calls/2010/02/softwarepreservation.aspx> accessed 4 April 2010.

5 For details of the full set, please refer to the Completion Report for this study, document number CC443D007-0.5 and dated 15 October 2010.

6 <http://creativecommons.org/licenses/by-sa/2.0/uk/> accessed 7 December 2010.

7 <http://www.e-science.stfc.ac.uk/projects/software-preservation/preserving-software.html> accessed 4 October 2010.

8 <http://www.oss-watch.ac.uk> accessed 4 October 2010.

9 <http://www.cog.org.uk/> accessed 4 October 2010.

1.3.3 This framework documents the key practical constructs (purposes, benefits, scenarios and approaches) uncovered and assimilated during this work. It is a synthesis of existing ideas and approaches illustrated with examples and case studies throughout. It is not intended to be a final and definitive answer, but a new step in the emerging practice of software preservation and sustainability.

1.4 Scope

1.4.1 The scope of this framework is broad and includes all types of software in UK Further and Higher Education (FE/HE). Such software includes 'rough and ready' code, agile developments and robustly engineered code, from different development environments, at different levels of the software stack (network, middleware, application) and for a whole range of purposes such as administration teaching and learning, research, *etc*. It includes non-licensed code not intended for release and closed code, but the focus is on open source software (and all licences therein). Whilst not explicitly about hardware, some software is hardware-dependent.

1.4.2 It should be noted that this is not an introduction to more general digital preservation. Nor does it set or advise on the organisational context for software preservation. Both of these are covered in detail elsewhere.[10] One of the study's key messages is that preservation should be considered in an integrated manner; so that if, for example, some data needs preserving then the software used to interpret/manage that data is not forgotten, or if a particular instance of some software runs counter to an organisation's overall preservation policy then the problem should be considered in the round.

1.5 Approach

1.5.1 The purposes and benefits given in this framework document are a synthesis from background materials, interviews with stakeholders and those identified in the case studies. An initial list of purposes and benefits was then tested and refined with developers at a Community Engagement Workshop in July 2010. This framework document contains the refined set of purposes and benefits.

1.5.2 It should be noted that there are many different ways one could organise and structure the purposes and benefits. We have presented one view in this document which is amenable to the wide range of audiences. Also, due to the paucity of robust research in this area we have relied on stated benefits, rather than demonstrable benefits. Each instantiation of a cost-benefit analysis, or business case, or benefits realisation plan, should, obviously, carefully consider and justify each benefit they assert.

1.6 Terminology

1.6.1 A tremendous range of terminology is used across the sector to describe similar concepts, practices, *etc,* in and around the longer-term aspects of software. To provide clarity to the reader of this report we have tried to be consistent in our terminology and to use commonly understood (if not preferred) terms.

1.6.2 In particular, we have chosen to use the following terms regarding software:

- **Maturity**: state of development and robustness of a particular software release; common stages of maturity include prototype, proof of concept; alpha, beta, pilot, and production;

[10] For a general introduction see, for example, the DPC's Digital Preservation Handbook <http://www.dpconline.org/advice/preservationhandbook/introduction> or the DCC's Curation Reference Manual <http://www.dcc.ac.uk/resources/curation-reference-manual>, both accessed 4 October 2010.

- **Maintenance**: the IEEE definition of maintenance is "The modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment".
- **Sustainability**: in some of the this study's outputs we have used this term as it is more familiar, and therefore appropriate, to the target audience than preservation; sub-section 2.3 explores the terms software preservation and software sustainability in detail.

1.7 Overview of this document

1.7.1 The rest of this report is set out as follows:

- Section 2 sets out the background to, and landscape of, software preservation;
- Section 3 provides a framework containing the purposes and benefits of software preservation and various scenarios where preservation is pertinent;
- Section 4 presents guidance to improve decision-making about software preservation;
- Annex A sets out different approaches to software preservation.

1.7.2 As this is a framework there is little in terms of a connective narrative. However, there are numerous case studies which relay real life stories of software sustainability and preservation.

This page is intentionally blank

2 Software preservation and sustainability

2.1 Introduction

2.1.1 This section sets out the background to, and landscape of, software preservation. The JISC-funded 'Significant Properties of Software' study[11], essentially the precursor to this study, produced an excellent introduction and framework to software preservation. This is highly recommended reading, particularly as it retains its relevance. To provide a background without wasteful duplication, material has been liberally drawn from this study. All quotes in this section, other than those indicated, are from this work.

2.2 What is software?

2.2.1 "Software is defined [on Wikipedia] as: 'a collection of computer programs, procedures and documentation that perform some task on a computer system.' Computer programs themselves are sequences of formal rules or instructions to a processor to enable it to execute a specific task or function… The term [software] is sometimes used in a broader context to describe any electronic media content which embodies expressions of ideas stored on film, tapes, records *etc* for recall and replay by some (typically but not always) electronic device... However, for the purposes of this study, such content is considered a data format for a different digital object type, and is thus out of scope of this study."[11]

2.2.2 "Software is a very large area with a huge variation in the nature and scale, with a spectrum including microcode, real-time control, operating systems, business systems, desktop applications, distributed systems, and expert systems, with an equally wide range of applications and also constraints of the business model from personally coded systems (typical in research), open-source systems, to commercial packages."[11] A recent trend has been for third party companies to deliver software functionality in a virtualised manner, for example so-called Software-as-a-Service (SaaS). In this instance, the architecture is such that the user of the software does not have access to the software itself. Approaches to preservation of SaaS require further research.

2.3 What is software preservation and sustainability?

2.3.1 Software preservation is intrinsically about reproducibility of functionality and results over time. "Software preservation [is] a term that was not necessarily considered a great deal, and when it [is], it means different things to different people."[11] We see two main cases:

- **Active or living preservation**, where software is continuing to be supported and maintained, and in addition to the preservation benefits there is also immediate and ongoing benefit from continued use;
- **'Classic' preservation**, where the software lies dormant (certainly without active development and releases, and potentially without use, support or maintenance) and the aim is to keep software intact for future use.

2.3.2 The former supports the latter as the longer that software is active the easier it is then to preserve. Though the term preservation is not in common use, many use the term sustainability which is closely aligned to active or living preservation, but where there is less focus on the need and benefits of preservation.

---

11 *The Significant Properties of Software: A Study*, Matthews et al, STFC, March 2008.

2.3.3 Most approaches to either type of software preservation do not guarantee perfect reproducibility – the fragility of software is generally too great. The 'Significant Properties of Software' study[11] therefore proposed the notion of adequacy[12] of preservation, to complement the notion of authenticity of preservation.[13] At the top-level, three levels of adequacy are given, namely that the preserved software:

- performs "exactly" as the original;[14]
- performs with small deviations from the original;
- performs only core functionality.

2.3.4 Preservation is for the long-term, but this time frame should be related to the purpose of preservation. As noted in the Open Archival Information Systems (OAIS) Reference Model (ISO 14721), when one talks about long-term preservation 'long-term' "is long enough to be concerned with the impacts of changing technologies, including support for new media and data formats, or with a changing user community". In the case of software, this might only be a few years.

2.3.5 As the 'Significant Properties of Software' study[11] sets out in detail, there are four aspects to software preservation:

- storing a copy of a software product;
- enabling its retrieval in the future;
- enabling its reconstruction in the future;
- enabling its execution in the future.

2.3.6 A significant element of this is enabling an understanding of the software in the future.

2.4 Is software preservation different from other kinds of preservation?

2.4.1 Software preservation is a particular type of digital preservation. It has seen less attention than data preservation and preservation of other digital objects, both in terms of research and in terms of practice. But is there a fundamental difference between software and other digital objects that are preserved? After all, software is a digital file. There is, with the rise of the Internet and dynamic web content, also a growing grey area in even distinguishing between software and data or content, for instance is an embedded Flash file better thought of as web content or software?

2.4.2 We believe that there are some notable distinctions, and these include:

- **All software is truly unique**: there are usually file formats for data and other digital objects, but all software differs massively;
- **Software is usually very complex**: data and other digital objects can be complicated but software offers often subtle behaviour that can be dependent on many conditions; understanding someone else's software is a difficult task and it does not translate well;
- **Software has more intricate and faster-changing dependencies**: the ability of software to compile or run, and the resulting behaviour, is dependent on many factors (*eg* system configurations) and these change easily (*eg* seemingly small system changes can result in non-functioning software); moreover, the technologies (system libraries,

12 The study stated that a software package (or indeed any digital object) can be said to perform adequately relative to a particular set of 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. By measuring the adequacy of the performance, we can thus determine how well the software has been preserved and replayed.

13 A preserved digital object can be said to be authentic if the object can be identified and assured to be the object as originally archived.

14 Including any undesired behaviour due to bugs.

languages, compilers, *etc*) are changing quickly and new generations of technology occur regularly. With no inherent 'backwards compatibility' complex and unique software very often ends up non-functional without software maintenance.

2.4.3 Despite these distinctions, guidance on general digital preservation still holds. For example, media obsolescence and recovery can be part of a software preservation project.

2.4.4 Perhaps the key reason for a different approach is that those who deal with software are often less aware of other preservation and curation activities, and curators and archivists are (generally) not familiar with software development.

2.5 Is software engineering the same as software preservation?

2.5.1 "It can also be observed that there is a large overlap between the requirements for software preservation and those of software engineering, especially for large software development which has a long lifetime in production and requires extensive adaptive maintenance. Both require the high-integrity storage, and replay of software. However, there are also significant differences."

2.5.2 "Software engineers are mainly concerned with maintaining the functionality of current systems in the face of software and hardware environment change, correcting errors and improving performance, and in [changing] functionality.[15] They will typically deprecate and eventually obsolete past versions of the software. They are much less concerned with maintaining reproducibility of past performance, which may be the concern of software archivists. So in general, software preservation is not what most software developers and maintainers do."

2.5.3 "Nevertheless, we argue that many of the approaches to software preservation mean in practice that the [practices] of software engineers are in fact appropriate to software preservation, and many of the tools, techniques and methodologies of software engineers are useful to software preservation, and good software preservation practice should adopt, adapt, and integrate these techniques. Indeed, a conclusion which arises from [the 'Significant Properties of Software'] study can be summarised as: **Good software preservation arises from good software engineering**."[11]

2.5.4 Software engineering is thus a different and wider topic, but does enable software preservation. Software engineering principles relevant to software preservation include:

- clear licensing;
- clear documentation;
- commonly adopted and modern programming language;
- modular design;
- clear revision management and change control;
- risk management;
- clearly established software testing regime and validated results;
- open and common standards;
- clear separation between data and code;
- clear understanding of dependencies.

15 The IEEE definition of maintenance is "The modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment". These different types are formally classified as Corrective maintenance (fixing faults), Adaptive maintenance (adapting to changes in environment), Perfective maintenance (meeting new/different user requirements), Preventative maintenance (increasing maintainability).

2.5.5 Use of significant properties[16] of software, as part of a systematic and thorough approach to documentation, is also recommended.

2.6 Is software preservation only relevant to research software?

2.6.1 Though the 'Significant Properties of Software' study[11] concentrated largely on mathematical, scientific and e-Science software (in order to limit the scope), software preservation potentially applies to all software in FE/HE. For example:

- **Learning and teaching domain**: *eg* preserving software learning objects for increased sharing;
- **Research software**: *eg* preserving software to retain a full record of research outputs or to enable an audit of research activities or of research-derived policy;
- **Administrative domain**: *eg* preserving software to retain audit capability for key records;
- **Office functionality**: *eg* the experience of using archived web material using the latest web browsers is very different from that of using the web browsers of the time; this indicates a need to preserve the browsers.

2.7 Does all software need to be preserved?

2.7.1 No. Not all software needs, or should, be preserved or sustained. As this benefits framework will demonstrate, some software offers great benefit if it were to be preserved. To other software, little such benefit could be ascribed – and given the costs of preservation – no case could be made. Our intention with the purposes, benefits and questions set out later is to help the reader make better decisions regarding preservation.

2.7.2 Not all artefacts associated with some software need to be preserved. "Software is inherently a complex object, composed of a number of different artefacts. At it simplest, a piece of software could be a single binary file; however, even in that case, it is unlikely to be standalone, but accompanied by documentation, such as installation guides, user manuals and tutorials. Further there may be test suites, specifications, bug-list and FAQs. More complete software packages will also include source code files, together with build and configuration scripts, possibly from a number of different systems and packages, with more complete documentation, including specifications and design documents (including diagrams) and Application Programming Interface (API) descriptions. Software will also have dependencies on a wider environment, including software libraries, operating system calls, and integration with other software packages, either for software construction, such as [development environments], compilers or build management systems, or in the execution environment, for example web-applications depending on web servers for execution and client browsers for user interaction. Thus a complete software preservation task may seek to preserve some or all of these artefacts, and, equally importantly, their dependencies upon each other."[11]

---

16 The JISC-funded InSPECT project website, <http://www.significantproperties.org.uk> accessed 7 October 2010, defines a useful description of significant properties: "Significant properties are those aspects of the digital object which must be preserved over time in order for the digital object to remain accessible and meaningful. An institution with curatorial responsibility for digital objects cannot assert or demonstrate the continued authenticity of those objects over time, or across transformation processes, unless it can identify, measure, and declare the specific properties on which that authenticity depends. Nor can it undertake the preservation actions required to maintain access to those objects, unless it can characterise their current technical representations with sufficient detail." The Significant Properties of Software study applied this concept to software.

2.8 What are the drivers and inhibitors for and against software preservation?

2.8.1 For the interested reader, and for context, the following table sets out illustrative drivers and inhibitors for software preservation:

PESTLE factor	Drivers	Inhibitors
Political	– 'Right to data' initiative supports data preservation and creates demand for software preservation – Major structural change in the sector likely to lead to loss of knowledge and responsibilities – Shared services agenda supports software sharing – Impact agenda may drive software reuse by creating an economic imperative to reuse	– Move to Commercial Off The Shelf (COTS), outsourcing, cloud and shared services reduces bespoke software development within institutions, meaning less software exists to preserve – Impact agenda may drive towards short-term benefits and away from long-term preservation
Economic	– Current funding cuts reinforce reuse arguments – Current funding cuts mean that capital expenditure is delayed, meaning existing software must be maintained for longer – Concentration on a few strategic priority areas (*eg* energy) changes balance of funding (benefiting some areas via greater infrastructure funding)	– Difficulties in finding new funding for software preservation – Current funding cuts harm preservation efforts – Concentration on a few strategic priority areas (*eg* energy) changes balance of funding (harming some areas via reduced infrastructure funding) – Benefits (and skills) are misaligned since (1) Software maintenance is intrinsically dull (for most developers) when compared to new developments; (2) Software developers aren't going to be around long-term anyway as they move projects; (3) Other parties benefit from reuse; (4) Librarians and archivists typically don't have the technical skills to preserve software – Software generally costs a lot to maintain – Difficulties in predicting (re)use means poor predictions of long-term value (and uncertainty discourages action, and uncertainty of value dissuades reuse) – Short funding horizons discourage reuse – Free-riding of openly shared software discourages contributions – Little economic incentive against 'prestige projects' that redevelop existing functionality rather than small projects that make reuse of existing software – Preserving all software would be unaffordable

Social	– Information society requires better information management which can be underpinned by long-term software and data preservation – Impact agenda may drive software reuse	– Impact agenda may drive towards short-term benefits – Much bespoke software and little culture of software reuse – Little culture of publishing and reusing software - little social incentive against 'prestige projects' that redevelop existing functionality (the 'not invented here' syndrome) – Many find it difficult to share imperfect code and do not want the burden of polishing code and supporting users – Lack of clear responsibility for software preservation issues – Researchers tend not to have software engineering skills
Technological	– Rise of virtualisation should make emulation easier – Trends towards standards (especially open standards) and commoditisation supports preservation efforts – Trends towards higher-level languages makes transferring knowledge about software easier	– Software is inherently fragile (sensitive to changes in environment) – Accelerating technological change changes software environment and dependencies ever faster – thus obsolescing software ever faster[17] – Accelerating technological complexity increases the preservation challenge – Move to SaaS means binaries and source codes are unavailable – Move to agile techniques reduces longevity of code
Legal	– Regulatory environment toughening (*eg* information retention requirements tightening)	– Preserving data in a secure way (*eg* for the Data Protection Act) is hard, especially so when support and maintenance for the underlying software is no longer available
Environmental	– Environmental regulations on hardware (especially disposal) may reduce throwaways	– Minimising energy overheads is an inhibitor to preserving indiscriminately

Further information and useful resources

The Significant Properties of Software: A Study
http://www.jisc.ac.uk/media/documents/programmes/preservation/spsoftware_report_redacted.pdf

Sustainable economics for a digital planet: Ensuring long term access to digital information
http://brtf.sdsc.edu

Journal of Software Maintenance and Evolution: Research and Practice
http://eu.wiley.com/WileyCDA/WileyTitle/productCd-SMR.html

Software Sustainability Institute
http://www.software.ac.uk/resources
http://www.slideshare.net/SoftwareSaved

Software Sustainability: Looking Past the Myths
http://www.omii.ac.uk/video/sustainability.jsp
http://www.slideshare.net/npch/software-sustainability-looking-past-the-myths

The Software Obsolescence Minefield (Component Obsolescence Group)
http://www.cog.org.uk

[17] Consider, for example, increasing frequent software releases, software in perpetual beta, the demise of 'gold release', shorter expected lifetimes and increasing trend towards disposability.

NSF Workshop on Cyber-Infrastructure Sustainability
http://cisoftwaresustainability.iu-pti.org/

Measuring Software Sustainability
http://doi.ieeecomputersociety.org/10.1109/ICSM.2003.1235455

JISC-funded resources on general sustainability
http://www.jisc.ac.uk/events/2010/07/jif10/virtualgoodybag/understandingsustainability.aspx
http://sca.jiscinvolve.org/wp/business-modelling-publications/

General digital preservation resources

http://www.dpconline.org/advice/preservationhandbook/introduction
http://www.dcc.ac.uk/resources/curation-reference-manual
http://jiscpowr.jiscinvolve.org/wp/guide/
http://blogs.ukoln.ac.uk/jisc-bgdp/

This page is intentionally blank

3 Purposes and benefits

3.1 Introduction

3.1.1 This section sets out the all-important purposes and benefits of software preservation. It provides the basis for the next section, where the purposes and benefits are used, along with other decision factors, to offer guidance on making better decisions about software preservation. Case studies are provided throughout this section to highlight real-life purposes and benefits, and also offer other people's experience and lessons on how to, and how not to, go about software preservation and sustainability.

3.1.2 A key challenge in digital preservation is being able to articulate, and ideally prove, the need for preservation. As the Blue Ribbon Task Force final report says "The first challenge to preservation arises when demand is diffuse or weakly articulated. Addressing the matter of demand is always the first step in developing sustainable preservation strategies… The value of digital assets is best understood as what digital materials are good for, and that is usually understood as the ways that the materials are used—to advance knowledge, entertain or bring pleasure, help solve problems, or inform public policy."[18] As there has been no prior attempt to identify and categorise these benefits systematically, JISC funded this study partly to do so. As a framework it is structured but extendable.

3.1.3 Our framework consists of purposes[19], benefits[20] and scenarios of software preservation:

- First, four (relatively orthogonal) purposes are identified. These are derived from assessing stated reasons for preserving software; each provides the essence of a rationale for software preservation. In reality, everyone should have their own purpose specific to them, their individual circumstances and the software. Such a real-life purpose may not be purely for software preservation – since there are other purposes with overlapping activities (*eg* aiming for openness). However, our four purposes strictly to do with software preservation are:
  - Encourage software reuse (sub-section 3.2);
  - Achieve legal compliance and accountability (sub-section 3.3);
  - Create heritage value (sub-section 3.4);
  - Enable continued access to data and services (sub-section 3.5).
- Second, a range of benefits are identified against each purpose. Due to a vast array of different types of software and scenarios in which software preservation might be considered there is a wide range of benefits. Only a few will apply in each case, but our framework aims to be as broad and inclusive as possible. This also matches the principles of the Blue Ribbon Task Force final report which says that "Each user community will identify its own set of values and benefits in the digital materials they demand" – therefore we do not attempt to describe these benefits definitively but list them for easy review, selection and tailoring by user communities.
- Finally, a range of scenarios are also identified for each purpose, to give some illustrative examples of where the purpose and accompanying benefits might be relevant.

---

18 *Sustainable economics for a digital planet: Ensuring long term access to digital information*, Final Report of the Blue Ribbon Task Force on Sustainable Digital Preservation and Access, February 2010, <http://brtf.sdsc.edu> accessed 5 October 2010. The report also says on the matter of benefits that "…well-articulated demand starts with a clear and compelling value proposition about the benefits to be gained by having, in our case, access to information [and presumably software] at some point in the future"

19 For which the OED definition is "the reason for which something is done or for which something exists".

20 For which the OED definition is "an advantage or profit gained from something".

3.1.4 Because there is potential misalignment between the benefits resulting from software preservation and benefits accruing to the developer or maintainer of the software who might be expected to contribute to software preservation, we also felt it important to present some 'personal benefits' to the developers too. These are given in sub-section 3.6.

3.2 Encourage software reuse

3.2.1 "The reuse of a software [artefact] is its integration into another context. The [reason for] reuse is to reduce cost, time, effort, and risk; and to increase productivity, quality, performance, and interoperability... The most common type of reuse is the reuse of software components, but other [artefacts] produced during the software development process can also be reused: system architectures, analysis models, design models, design patterns, database schemas, web services, *etc.*"[21]

3.2.2 "Perhaps the prime motivation to preserve software for most people is to save effort in recoding. Code from the past 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 most existing software repositories, and a significant part of the effort which is undertaken by software developers both in-house to end-user organisation, and also within software houses. 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. As a consequence, an important source of information on [the] significant properties [of software] for preservation is the best practice on software maintenance and reuse, a long recognised part of good software engineering. 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 environment and provide the correct functionality."[22]

3.2.3 Possible scenarios for reuse include:

- Continuing operational use in institution;
- Increasing uptake elsewhere;
- Promoting good software.

3.2.4 The benefits include one or more of the following covering software and resulting systems:

- Reduced development cost;
- Reduced development risk;
- Accelerated development;
- Increased quality and dependability;
- Focused use of specialists;
- Standards compliance;
- Reduced duplication;
- Learning from others;
- Opportunities for commercialisation.

---

21 <http://www.esdswg.com/softwarereuse/Resources/library/reuse-definitions/> accessed 8 October 2010. This resource goes onto to say that the following does not count as reuse "(1) Software developed and used repeatedly by the same people on the same project is regarded as "good programming practice" and is not typically counted as reuse; (2) Product maintenance and new product versions; we do not usually claim the base code as reuse; (3) Use of operating systems, database management systems, and other system tools is generally not regarded as reuse; (4) The use of commercial off-the-shelf (COTS) software and its open source equivalents is generally not regarded as reuse."

22 *The Significant Properties of Software: A Study*, Matthews et al, STFC, March 2008.

3.2.5 These benefits accrue to the organisation hosting the software development and reusing software, but software reuse will also make life easier for the developer. These benefits can offer hard 'savings' to an organisation – the financial return on software reuse is well explored by the literature.[23] The extent of the overall benefit depends on the scale and complexity of the reused software and, importantly its quality. For example, development risk can actually be increased if reused code is poor, isn't modular, uses global parameters, *etc*.

3.2.6 The end benefits are perhaps best summarised as:

- Greater efficiency;
- Increase flexibility and responsiveness;
- Increased community participation.

Re-usable Educational Software Library (RESL) (www.resl.ac.uk)

RESL is an online database of resources centred around re-using educational software. RESL was developed as part of a project entitled Software Use, Re-use & Customisation in Education (SoURCE).[24] SoURCE was run by the Open University with partners the University of Wales at Bangor, De Montfort University and Middlesex University. SoURCE ran from September 1998 to December 2001. Its lessons and findings are written up in a final report.[25]

RESL "set out to investigate the feasibility of a national re-usable educational software library to provide access to software resources, guidelines and other materials relevant to the adoption and adaptation of educational software." Whilst originally intended to include mostly software it ended up having "little software compared to case studies about using software... This is because participants from across UK HE have indicated this is more useful. Software dates quickly, and is often hard to re-use. However, peoples' experience in trying to re-use it is valuable and transferable. Hence [case-studies, articles, reports *etc*] of good practice make up most of RESL's content."

Some of the lessons identified were:[26,27,28]

- One of the key benefits of academics customising software is that it encourages them to reflect on their educational practice.
- Decisions taken early in the lifecycle of software development have a profound impact on its reusability and range of contexts for use.
- The experience of customisation in SoURCE suggests that decisions to develop re-usable software are only taken under fairly exceptional circumstances.
- The main barriers to software customisation and reuse appear to be cultural at both individual and institutional levels.
- The greatest opportunity for, and cultural acceptance of, reuse seems to occur when content-specific objects at a very low level of granularity can be identified and fitted flexibly into the curriculum.

23 See, for example, *The business case for software reuse*, Poulin et al, IBM Systems Journal, Vol 32, No 4, 1993.

24 <http://www.source.ac.uk> accessed 4 October 2010.

25 SoURCE Evaluation Report, Beetham et al, MET-DEL-2, August 2001.

26 <http://www.source.ac.uk/software_development.htm> accessed 4 October 2010.

27 <http://www.source.ac.uk/customisation_&_reuse.htm> accessed 4 October 2010.

28 Note that whilst the SoURCE project began over a decade ago it is surprising how many of the findings and lessons appear to be as relevant today.

- Once a decision to customise and reuse has been taken there are often many pragmatic barriers to be overcome at the site of reuse. In addition to problems involving the software itself, including lack of appropriate facilities, lack of technical support, integration with existing technical systems *etc*, there are generic barriers to any change in the mode of delivery of learning. These include inflexible timetables, inflexible teaching facilities and lack of time to undertake curriculum development.

There does not appear to have been any resources added to RESL after 2002.

Further information and useful resources

NASA's approach to software reuse
http://softwarereuse.nasa.gov/
http://www.esdswg.com/softwarereuse/Resources/library/reuse-definitions/
http://www.esdswg.com/softwarereuse/Resources/rrls/

3.3 Achieve legal compliance and accountability

3.3.1 Software preservation can be necessary to achieve legal compliance and accountability. A greater use and reliance on information has led to new laws and regulations that organisations must abide by. Minimising the burden of compliance is key to freeing up time and money to focus on an organisation's 'real business'. Some possible scenarios where software may need to be preserved for compliance or accountability reasons include:

- Maintaining records or audit trail;
- Demonstrating integrity and authenticity of data and systems;
- Addressing specific contractual requirements;
- Addressing specific regulatory requirements;
- Resolving copyright or patent disputes;
- Addressing the need to revert back to earlier versions due to IP settlements;
- Publishing research openly for transparency;
- Publishing research openly as a condition of funding.

3.3.2 Legal compliance is mandatory and the benefits of preserving software in this context should be self-evident, but would include:

- Reduced exposure to legal risks;
- Avoidance of liability actions;
- Easily demonstrable compliance lessens audit burden;
- Improved institutional governance;
- Enhanced reputation.

3.3.3 Accountability is more subjective and variable than legal compliance, but the benefits of preserving software in this context include:

- Social expectations met;
- Sense of responsibility;
- Demonstrable leadership.

3.3.4 Most of these benefits accrue to the organisation concerned; though sometimes the society benefits (*eg* widespread accountability of research or other public funding benefits the public).

Legal and regulatory requirements in the aerospace industry[29]

"The objective of LOTAR International is to develop an auditable process for the long-term archiving (LTA) of digital data, *eg* 3D CAD and Product Data Management (PDM) data... The LOng Term ARchiving (LOTAR) project [is necessary because of] the legal and business requirements … within the aerospace industry.

A general demand for long term archiving [of] all legal and certification relevant documents is a result of [the] Aircraft Certification requirements of Authorities (Joint Aviation Requirements (JAR)[30], Federal Aviation Administration (FAA) and others), national laws and legal practice concerning with product liability and guarantee. Therefore basic requirements independent from 2D or 3D product documentation are:

- to ensure continued readability, authenticity and identity of the records;
- to demonstrate to the authorities proper functioning of the records system, and;
- to maintain the capability to retrieve type design data in a usable form over the validity period of the Type Certificate.

The life cycle of applications and storage technologies has to be considered by setting up a long term archiving and retrieval standard. Approximately every three years a change in the application technology happens, for the technology of storage and retrieval this is every ten years. In comparison with an archiving period of fifty up to one hundred years in the aerospace industry, the technology life cycle plays a major role... Besides the challenges caused by different technology life cycles the risk of data migration has to be considered. The use of a native CAD-format may lead to wrong or even no results when loading in a new generation of CAD-Systems."

With CAD in aerospace applications it has proved hard to separate data from software, as modern CAD software does not just passively display drawings. Instead, the software and the data together provide a model that is active and can be manipulated and queried to draw out behaviour of the model. This means that LOTAR validates after data is read by the software. The coupling between data and software is exacerbated by the proprietary and closed file format of the predominant CAD software.

The legal and regulatory requirements that demand some form of software preservation come from the mix of short and very long timeframes involved. The time between CAD versions can be only six months, and the life of a CAD system is ten years. This can be compared [to] the life of the product which is seventy years or more (*eg* 30 years of production, followed by many more decades of servicing, spares and modifications for such a long lifespan). So whilst the CAD system will be obsolete after ten years, and probably forgotten after twenty years, the legal liability goes on and on.

Some of the relevant tenets from the LOTAR project are:

- sustaining models not drawings;
- model is data plus algorithms;
- preservation planning is about the governance.

29 All quotes taken from <http://www.prostep.org/en/project-groups/long-term-archiving-lotar.html> accessed 5 October 2010, supplemented with notes taken from a LOTAR presentation at the DPC event "Designed to Last" on 16 July 2010 (<http://www.dpconline.org/events/designed-to-last-preserving-computer-aided-design.html> accessed 20 July 2010).

30 "The Civil Aviation Authorities of certain European countries have agreed common comprehensive and detailed aviation requirements (referred to as the Joint Aviation Requirements (JAR) with a view to minimising Type Certification problems on joint ventures, and also to facilitate the export and import of aviation products. The JAR are recognised by the Civil Aviation Authorities of participating countries as an acceptable basis for showing compliance with their national airworthiness codes." <http://en.wikipedia.org/wiki/Joint_Aviation_Requirements> accessed 5 October 2010.

Software and records preservation for Moodle[31]

Context

The following is an extract from a news story on the UCL website :

"[The] inquiry panel was set up in January 2010 following the arrest of Mr Abdulmutallab in the US on 25 December 2009 on suspicion of attempting to bomb a US civil aircraft, and the subsequent criminal charges brought against him.

The inquiry panel was asked to explore the nature of Mr Abdulmutallab's experience as an undergraduate student of UCL between 2005 and 2008, including his period as President of the student Islamic Society.

Sir Stephen Wall, Chair of UCL Council, said: "Given the seriousness of the charges against him, UCL announced earlier this year it would be establishing a panel to explore the nature of Mr Abdulmutallab's experience as an undergraduate student of UCL, investigate whether there were at UCL at that time conditions that might have led to Mr Abdulmutallab's engaging in acts of terrorism, and whether there are at UCL today conditions that might facilitate the possibility of other students doing so in future.

'The panel collected evidence from across the institution, and interviewed a wide range of members of the UCL community who were well placed to offer insights on the issues addressed. We welcome the central conclusion that there is no evidence to suggest either that Umar Farouk Abdulmutallab was radicalised while a student at UCL or that conditions at UCL during that time or subsequently were or are conducive to the radicalisation of students.'"

Software preservation

UCL uses Moodle for its virtual learning environment. Moodle at UCL runs on Apache web server running PHP and uses a MySQL database. It provides many features such as the uploading of assignments, forums, chat and blogs. The Learning Technology and Support team were not approached to provide details of Mr Abdulmutallab's Moodle activities but the requirement to be able to access Moodle data for students who had left UCL became apparent at this time.

This poses some software preservation issues. Moodle data is saved in a MySQL database. The database and associated datafiles (*eg* files related to assignments) are backed up nightly. One of the team says "If we were asked to retrieve all the data for student we would need to restore the database to a MySQL server, and Moodle on to an Apache web server. Since MySQL is upgraded periodically we can't guarantee that an older database will be readable on newer versions of MySQL. Likewise PHP on the Apache web server will inevitably have been upgraded and older versions of Moodle may not be compatible with new versions of PHP." To address these issues UCL is now putting together a proposal to archive a read-only instance of Moodle on a virtual machine.

Clear Climate Code initiative[32]

Context

"ccc-gistemp is a software project started in 2008 by Nick Barnes and David Jones. It is a reimplementation of GISTEMP, a piece of software developed by NASA Goddard Institute for Space Studies (GISS) that produces an estimate of historical global average temperature trends.

31 Personal communication with Jo Matthews at UCL.

32 Case study contributed by David Jones (Ravenbrook and Clear Climate Code).

NASA GISS published the source code to GISTEMP in 2007, but it was found to be too obscure to be of much public benefit. Clear Climate Code was founded and we started the ccc-gistemp to address that problem (Clear Climate Code has now gone on to be an activity of the new Climate Code Foundation). As software engineers we could improve the published GISTEMP code and thereby it's public benefit (in particular pointless debates about unclear bits of Fortran were disrupting the public discussion of policy issues).

Aim and plan

Our aim in making ccc-gistemp was to make the implementation of the algorithm (described in various peer-reviewed published papers) as clear as possible. We also wanted to the results of ccc-gistemp to be as close as possible to GISTEMP so that there was no doubt that ccc-gistemp and GISTEMP implemented the same algorithm (hopefully this allows the discussion to move forward and consider the algorithm itself rather than an unclear implementation of it). The GISTEMP code as is certainly fit for the purpose for which it was created: implementing a calculation in support of a series of scientific papers. However, this code came, by accident or otherwise, to gain a prominent place in public debate, and was certainly not fit for such scrutiny (what code produced under publishing deadline circumstances over a period of 30 years would be?).

We decided that the best way to proceed would be a line-by-line rewrite of the original GISTEMP code. We chose to do the new implementation in the programming language Python. Some of the reasons we chose Python were: clarity, familiarity, and longevity.

Programming for reproducibility

In programming for reproducibility we were quite fortunate. The GISTEMP algorithm was divided into a small number (six) of serial steps, and the steps communicated via intermediate files. Each step would take the output of the previous step (in a small number of files), process the data, and produce a new set of intermediate files. The steps were variously written in bits of shell, C, Fortran, and Python. Our initial work proceeded by taking a step and rewriting all of it in Python while maintaining the same inputs and outputs. Because of this arrangement, we didn't have to attack any particular step first, and different people could be working on different steps at the same time. During this phase of development (until we had completely rewritten the program in Python) we had a system where we could run any step using either the original GISTEMP code for that step, or our replacement code where we had written it.

We attacked the larger more complex steps first, in order to give us some idea of the hard problems we would face and to minimise risk ('tackle the largest risks early'). The files handled and produced by each step were usually some novel Fortran binary format (or sometimes some novel text format), and we had to write modules to handle these files in Python. Python proved remarkably flexible, it was straightforward to handle binary Fortran files in Python (which we had identified as one of the larger risks).

Some of the steps in the GISTEMP code were 'integer to integer' steps in that they took large ensembles of integer data (weather station records) and produced large ensembles of integer data. By carefully reproducing various Fortran rounding algorithms (which we reverse engineered from the input and output data!) we were able to recreate these steps exactly. Other steps processed floating point data. We could not reproduce the output of these steps exactly as getting 'all bits exactly the same' would mean matching the exact precision (32-bit versus 64-bit versus internal 80-bit) for the operands and matching the exact order of floating point operations. No Fortran compiler guarantees these things, so it would be hopeless to try and replicate it. The best we can hope for is that we've implemented the same algorithm and the results are consistent with variations in floating point calculations (this is quite a tricky area). We had to write tools to compare files in the novel formats, for example to compare two gridded datasets to check that the differences were as small as we would expect.

Once we had got to the point where we had an all Python version and each step had outputs that matched (or matched as closely as was reasonably possible) the GISTEMP code, then we proceeded to simplify the Python code, and this started with removing lots of unnecessary rounding (to integer, for file output), and removing the passing of data via files. We still retain the ability to write intermediate files, but the computation now passes data internally.

Source code management

In order to bring in more community involvement the project we moved the source code repository from an internal Perforce repository at Ravenbrook to a public Subversion repository on Googlecode. We did not take very great care in selecting Googlecode as a repository, but it turns out to have many useful features. Because the Subversion repository is itself public, anyone can access any revision of the source code, and see every change that is made. This means we do not need to make formal releases so frequently, as anyone who was more than a little interested could access the public repository (though we do try and make frequent releases, having made five releases in 2010).

Another benefit of Googlecode is that someone else is doing the maintenance, and user-access-control is relatively simple. We don't have to look after the servers, and we don't have a special system for managing users. External contributors are identified and managed using their Google identities. No doubt other systems (github springs to mind) offer similar advantages. The bottom line is that existing online systems that are free to use are adequate, and in many cases better, then commercial systems that we would use in house.

Use of the code

GISS have said they want to use our new code, but this has not yet happened. Possible causes are the differences in training and the two groups respective "favourite toolboxes". ccc-gistemp is very much a fairly modern sort of Open Source project using a hip new language.

The corresponding GISTEMP project at GISS occupies only a small amount of their effort (10% of one FTE), and the group as whole is heavily invested in traditional large scale scientific programming of complex physical model simulations using specialised hardware (IBM AIX mainframes) and industrial quality commercial Fortran compilers. The impedance mismatch is considerable, but we hope to bridge it.

Lessons

- Understand the scale and complexity of the task by tackling the biggest risks first;
- Perfect reproducibility may well be impossible for some applications;
- Have a clear aim that gives a range of benefits; from encouraging software reuse, through improved maintainability and longevity, to the public policy benefits arising from scrutiny and transparency;
- A complete rewrite in another programming language is probably at least as expensive as the original software: the closer a degree of reproduction you desire (and we went for quite a high degree), the more expensive any preservation / emulation will be."

Further information and useful resources

Records management infoKit
http://www.jiscinfonet.ac.uk/InfoKits/records-management

Legal Guidance for ICT Use in Education, Research and External Engagement
http://www.jisclegal.ac.uk

3.4 Create heritage value

3.4.1 Software preservation can create heritage value, because software can be culturally, aesthetically, historically, and politically significant. There is some intrinsic benefit from preserving software that offers this significance. Whilst difficult to articulate this could be expressed as enabling a greater understanding of culture and history, learning from past mistakes, *etc*. These benefits accrue mostly to society, and are generally non-financial.[33]

3.4.2 Some possible scenarios for this purpose include:

- Ensuring a complete record of research outputs where software is an intermediate or final output;
- Preserving computing capabilities (software with or without hardware) that is considered to have intrinsic value;
- Supporting the work of museums and archives.

3.4.3 The 'Significant Properties of Software' study recognised this purpose and commented: "A small but significant constituency of software preservation is those museums and archives which specialise on preserving aspects of the history of computing and its influence on the wider course of events. These institutions thus want to preserve important software artefacts as they were developed at the time of their creation or use, so that future generations of historians of science (and the general public) can study and appreciate the computers available [at] that particular period, and trace its development over time."

Cultural significance and the preservation of Digital Games

Loughborough University's Department of Information Science published a paper in 2008 entitled "The Barriers to the Preservation of Digital Games: Questions on Cultural Significance".[34] It concluded that "Digital media is changing many aspects of our lives and digital games, with their position as a lead technology and the influence they have had on computing technology and other media. Yet, as part of our every-day lives, they have been overlooked as a valuable aspect of our cultural heritage and their preservation has received little attention in the literature on digital preservation. Despite this, their continued growth in popularity and an ever-increasing interest from academia suggest that they should be recognized as 'something with a history worth preserving and a culture worth studying"'. Offering a view on the challenges of preservation, it states that "Emulation, which is seen as the heart of software preservation, is the approach most often taken by games enthusiasts [...] Nevertheless, these activities are unstable forms of preservation because they are individual initiatives without long-term support".

Sir Salman Rushdie's archive at Emory's Manuscript, Archives, and Rare Book Library[35]

Emory University houses Sir Salman Rushdie's archive: "The celebrated writer's computer files, private journals, notebooks, photographs and manuscripts provide insight into his creative process, campaigns for human rights and celebrity."

33 Expressing and measuring the 'value of culture' is a deep and varied research topic. For a flavour see *Capturing the Public Value of Heritage*, The Proceedings of the London Conference, 25-26 January 2006.

34 <https://dspace.lboro.ac.uk/dspace-jspui/handle/2134/4988> accessed 29 September 2010.

35 <http://www.emory.edu/home/academics/libraries/salman-rushdie.html> accessed 5 July 2010.

The groundbreaking story of the archive, and its software preservation activities due to the donated computers, has been documented by the archiving team.[36] The archive contains both paper and electronic material: "the archive is a hybrid, meaning that [Emory] received not only one hundred linear feet of his paper material, including diaries, notebooks, library books, first-edition novels, notes scribbled on napkins, but also forty thousand files and eighteen gigabytes of data on a Mac desktop, three Mac laptops, and an external hard drive." This presented new challenges as the archiving team did not want just to preserve the data. Instead they wanted to preserve the experience of the writer's environment. "[Erika Farr, the Emory libraries' director of born-digital initiatives] is a big believer in preserving the whole ecosystem, or 'biostructure', of the author's digital archive: the hardware, software, programs, and applications, all the files and file names, search histories—even the order in which everything was installed. 'There is something fundamentally interesting about the computers themselves,' she says, 'as the medium between the user and the digital media.' … [Peter] Hornsby, who extracted the data from Rushdie's hard drives, felt it was crucial to emulate the author's working environment, creating a perfect duplicate that researchers could explore while safeguarding the original: 'The imprint of the writer's personality,' he says, 'lies within his computer.'"

Emory is providing multiple points of access into Rushdie's digital archive, including emulations of Rushdie's computers and searchable databases of files pulled off of his computers. An Open Source PowerMac Emulator called SheepShaver has been used for this, so that experience of using the terminals is as close to using Rushdie's computers as possible, including the now obsolescent word processors and even the desktop games.

Preserving Virtual Worlds for cultural reasons[37]

"Interactive media are highly complex and at high risk for loss as technologies rapidly become obsolete. The Preserving Virtual Worlds project […] explore[d] methods for preserving digital games and interactive fiction." The project was undertaken by a partnership of US universities led by the University of Illinois at Urbana-Champaign and was a National Digital Information Infrastructure Preservation Program project administered by the Library of Congress.[38]

The final report[39] highlights the cultural significance of virtual worlds: "As software artifacts exhibiting complex dependencies on platform, operating system, and network environment, virtual worlds are undoubtedly among the most imperilled forms of interactive digital content. As communities—shared spaces and places—they are defined by no less delicate and idiosyncratic skeins of people, relationships, memories, and folklore akin to those found within oral cultures. Virtual worlds are not virtually—that is, "almost"—real. They are instead, to borrow a phrase from Jesper Juul (2005), precisely half-real: they are human products, scripted and engineered out of millions of lines of code written by dozens or hundreds or even thousands of individuals, but they are also focalizers for powerful collective acts of the imagination that rely on the same willing suspension of disbelief that characterizes immersion in other media, like novels and films."

The final report details the challenges to preservation of virtual worlds, which were summarised as:

"**Hardware obsolescence** – The original console or computing platform used to run the game may cease to be supported or even available in the aftermarket.

36 <http://www.emory.edu/EMORY_MAGAZINE/2010/winter/authors.html> accessed 1 October 2010.

37 <http://pvw.illinois.edu/pvw/> accessed 30 September 2010.

38 Project partners are the University of Illinois at Urbana-Champaign (lead), the University of Maryland, Stanford University, Rochester Institute of Technology and Linden Lab. Second Life content participants include Life to the Second Power, Democracy Island and the International Spaceflight Museum. The Preserving Virtual Worlds project is funded by the Preserving Creative America initiative under the National Digital Information Infrastructure Preservation Program (NDIIPP) administered by the Library of Congress.

39 <http://hdl.handle.net/2142/17097> accessed 30 September 2010.

Software obsolescence – The original software needed to run the game - operating system, drivers, frameworks - may lose support, cease development, or become incapable of running on future hardware/software configurations.

Scarcity – Some video games are produced in limited quantities and are subject to the dangers of media decay. This is especially likely to be the case for special editions and releases, recalled games, or art games.

Third party dependencies – Currently most emulators are developed by the game community and are of questionable legality. They are also typically created without the benefit of the original specifications and are themselves at risk of becoming obsolete.

Complex, proprietary code – And an associated lack of documentation. Digital games are generally released as compiled binaries with no documentation of the compiling process, or even the programming languages used. Not having access to the source code or language specifications makes migrating or emulating software far more difficult.

Authenticity – The elephant in the digital preservation room, proving that a digital object is what it claims to be, free from tampering or corruption. Digital games enjoy many versions between the first prototype, the official release (on multiple platforms), and cracked or otherwise altered unauthorized editions. Especially for older games, the only extant copy may exist in a fan-run web repository, making the authenticity impossible to establish.

Intellectual Property Rights – The game development industry is highly creative and competitive, leading developers to be conservative with their intellectual property. Most have instituted restrictive shrink-wrap licenses reflecting this. And yet, once a game is no longer actively marketable, they are less likely to respond to inquiries about licensing for it.

Significant properties – What are the significant properties of a game that must be maintained with each transformation/preservation action? What makes Mario Mario? How important are font size and colour palette? What about the speed of text scrolling or sprite movement? What about controllers? How faithful must we stay to the original code? Significant properties are essential to define, as they play a major role in determining authenticity.

Context – Although not an immediate threat to the preservation of games, building contextuality is important to creating understanding for future users. This is truer for digital games than many other record types because, as technology advances, game players who have only been exposed to the latest and greatest may be apt to play an older game and say, "so what?" even though the game might have been revolutionary for its time. For example, Sierra's *Mystery House* was the first text adventure to incorporate graphics. An amazing breakthrough in its day, it seems crude in comparison to today's virtual environments."

The final report also has eight archiving case studies covering early video games, electronic literature and Second Life, the interactive multiplayer game.

Stephen M. Cabrinety Collection in the History of Microcomputing[40]

"The Cabrinety Collection on the History of Microcomputing [at Stanford University] is a collection of commercially available computer hardware, software, realia and ephemera, and printed materials documenting the emergence of the microcomputer in the late 1970s until 1995. Specifically, the collection documents the rise of computer games, with a focus on games for Atari, Commodore, Amiga, Sega, Nintendo, and Apple systems. As such, the software collection documents the increased technical ability of computer software programmers and the growing sophistication of computer-generated graphics from the early days of games like Pong to the current era of game systems like Nintendo 64."

Further information and useful resources

The Specimen Case and the Garden: Preserving Complex Digital Objects, Sustaining Digital Projects
http://dh2010.cch.kcl.ac.uk/academic-programme/abstracts/papers/html/ab-626.html

Preserving Digital Worlds
http://pvw.illinois.edu/pvwSoftware carpentry - wide range of advice for developing software research

3.5 Enable continued access to data and services

3.5.1 Preserving software can enable continued access to data and services. Without preservation, software data can be 'locked up' and inaccessible, and services must be discontinued due to obsolete software.

- **Continued access to data**: Software is used to create, interpret, present or otherwise manipulate and manage data and other digital objects. In this sense, software underpins data and other digital objects. Software preservation should be a consideration where the software cannot easily be separated from the data or digital objects. For instance, preservation of data (documents, images, *etc*) can require the preservation of format processing and rendering software in order to make the content accessible to future users. Ideally the two can be separated and the data or digital objects can be preserved independently of and without the software.[41] However, sometimes the two are more tightly coupled; for instance if the software and data come together to form an integrated model so the data by itself is meaningless, or if data in its raw form isn't in an open, human readable format. Where this is the case, it is necessary to preserve the software as well, since preserving the data but not the software makes very little sense.
- **Continued access to services**: Software systems underpin services too, by being part of a process that a service provider uses to engage with users. Software preservation is a consideration where otherwise that service could not operate.

3.5.2 The 'Significant Properties of Software' study identified this purpose and commented: "[Sometimes] it is necessary to preserve software 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 *per se*, so it may not be necessary […] to […] ensure that that software is reproduced in its exact form, but only sufficient to process the target data."

40 <http://library.stanford.edu/depts/hasrg/histsci/index.htm> accessed 5 July 2010.

41 If possible, turn a software preservation problem into a data preservation problem, as these are generally easier to handle. Data migration is invariably easier than software preservation.

3.5.3 Software preservation for enabling continued access to data and services is a potential issue across further and higher education. For instance, it is applicable to research data, business intelligence from corporate data, and learning objects for learning and teaching.

3.5.4 Some possible scenarios where this purpose might be relevant include:

- Reproducing and verifying research results;
- Repeating and verifying research results (using the same or similar setup);
- Reanalysing data in the light of new theories;
- Reusing data in combination with future data;
- 'Squeezing' additional value from data;
- Verifying data integrity;
- Identifying new use cases from new questions;
- Maintaining legacy systems (including hardware);
- Ensuring business continuity;
- Avoiding software obsolescence;
- Supporting forensics analysis (*eg* for security or data protection purposes);
- Tracking down errors in results arising from flawed analysis.

3.5.5 The benefits resulting from software preservation around research data and business intelligence include:

- Fewer unintentional errors due to increased scrutiny;
- Reduced deliberate research fraud;
- New insight and knowledge;
- Increased assurance in results.

3.5.6 The key end benefits from these benefits are improved research outcomes and greater efficiency.

3.5.7 The benefits resulting from software preservation around systems and services include:

- Current operations maintained;
- Opportunity for improved operations via corrective maintenance;
- Reduced vendor lock-in;[42]
- Improved disaster recovery response;
- Increased organisational resilience;
- Increased reliability.

3.5.8 The last four of these benefits are indirect. The key end benefit from the set of benefits is 'reduced operational and strategic risk' to the organisation. This may be seen as mainly risk mitigation, but it could offer competitive advantage (*eg* customers may value reliability).

[42] In particular this benefit stems from being able to keep older software (and hardware) running and thus being able to avoid/delay upgrade cycles; and from opening up source code to the community which can reduce dependence on a vendor.

OU Knowledge Network – 10 years of a successful repository service[43]

The Knowledge Network is a web-based information system that allows OU staff 'to find and share OU expertise about teaching and learning'. After being developed in a three year project back in 2000 it has seen continued growth and impact. Changes in the way it can be supported eventually forced a decision on if and how to sustain the service and preserve the valuable knowledge within. After a formal options assessment the service is being migrated to a new platform to be delivered and supported by a central systems team within the OU as part of a new enterprise content system. The original software developer has been involved throughout and "felt ownership of the service", and is now helping it to be 'cultivated' by the new team.

Early life

The system was developed in 1999/2000 as part of a £200k HEFCE-funded project on knowledge sharing and management (UNLOCK Project, Josie Taylor and Patrick McAndrew). The design of the application was developed by Doug Clow, James Aczel and Will Woods, with Alex Little doing the bulk of the programming. The service was launched in 2000 and subsequently run by the Learning & Teaching Technologies Team. It was a successful project with over 50% of the intended target audience using it. Such success has been attributed to the fundamental appeal of the service, and the lightweight metadata policy offering a 'low hurdle' for participation and the good access controls that separated early draft versions of content from content intended for wide release.

Mid-life

The project funding lasted for 3 years and so external funding stopped in 2003. After 2003, there were still lots of ideas of how to develop the service further, but less software developer effort to do this. Using an internal budget a post was funded until 2009 to:

- support users;

- develop the service based on user feedback;

- champion the service within a 'departmental evangelist' model.

In 2005 the OU started to look at an enterprise-level content management system. Through a tendering process it procured such a system that would be deployed and operated as a centrally run managed service. However the enterprise system was not immediately compatible with the Knowledge Network service: the Knowledge Network is an Adobe ColdFusion based platform, whereas the enterprise system was going to be a Drupal PHP-based web system with EMC Documentum managing the enterprise content and document life-cycle management. Also, part of the enterprise approach was to rationalise the number of supported platforms and so the support overheads of the Knowledge Network would run counter to this ethos. At the same time the Learning & Teaching Technologies Team lost some key support personnel and it became clear there was a reliance on specific skills that required frequent refresh.

In 2008, further development to the Knowledge Network was stopped, although the service was still available to use unsupported (*ie* 'as is') and still considered very successful, with:

- 16000 distinct users;

- 11000 active users accessing content;

- 2000 document requests a month;

- 7000 distinct documents and 5000 workspace pages.

[43] With special thanks to Will Woods who contributed to this case study in August 2010.

Deciding the future

Since the Knowledge Network represented "institutional information that we didn't want to lose" the decision was made to migrate the Knowledge Network over to the enterprise system. This followed a mid-2009 options appraisal that considered the following options:

1 - Migration onto new platform (providing some functionality, plus all the content);

2 - No migration and gradually phase the service out;

3 - Ground-up redevelopment of the service in PHP;

4 - Migrate content only.

Migration onto the new platform was presented as the preferred option, and demonstrated the importance of the functionality, and not just the content. Since the enterprise system had not gone live, the migration of a familiar system first was thought to help with getting buy-in for the enterprise system. The fallback in that option was to make the content searchable and taggable in the enterprise system if the functionality couldn't be implemented. Ground-up redevelopment of the service was considered risky as the bespoke code would lead to support costs and the central enterprise team may not be willing to support it, particularly if the code doesn't integrate with the enterprise system.

A new-life

Migration thus far has consisted of working with the central enterprise system team to prioritise Knowledge Network requirements and to map the current functionality over to PHP equivalents. Much of the current service is bespoke code since, in 2000, repository, collaboration, commenting, *etc* functionality was not available 'off the shelf'. Fortunately nowadays it is, and almost all of the high-priority functionality maps across to standard Drupal components or modules. The medium-level functionality can be implemented through customising Drupal modules. This means that the migration project should not be a massive development project. With the functionality planned to be ready in early 2011, content migration will then take place. User 'orientation' time has also been planned in order to win users' 'hearts and minds' for the migration. Eventually the team are looking to switch off the Knowledge Network altogether in July 2011. Thought is also being put into ensuring the high visibility of KN content (for example ranked highly in Google searches *etc*.) can be managed across to ensure that the external prominence of the information is maintained.

Lessons identified

Always plan for sustainability: always plan that services will continue to exist after funding stops since an institution can't just turn popular services off. Considering the scenario in 5-6 years time should help.

Start sustainability planning early: start the process a lot earlier than you would think. The team's planning has been informed by the experience of migrating an online survey tool, that it takes 18-24 months to do this properly.

Enforced technical preservation of an atmospheric model reduced research integrity

An atmospheric model was developed by an institution and ran on an institutional High-Performance Computer (HPC) resource. However, an upgrade to the HPC resource meant that the software no longer ran. Instead the team continued to maintain the old hardware for six months. As a precaution they took an archive of the entire disk to allow recovery.

Eventually the model's code was adapted so that it runs on the new hardware. However it doesn't run in all configurations and the team can't guarantee its integrity. At least one similar model in another organisation faced the same problem, but here the large up-front cost of migration to a new HPC platform was deemed affordable and the outcome much more desirable.

Long-term migration of a research critical aphid database[44]

Introduction

Back in the early 1960s a group of entomologists wanted to monitor moth populations using traps distributed all over the UK. They started doing this and extended data collection to aphids in the 1970s. Over time the research has been expanded further, for instance to include EU data. A key research finding is that aphids are a good indicator of climate change and so specific parameters of aphid populations are now used to track climate change. The research has also had an economic and social benefit in enabling the Aphid Bulletin System that is used to notify farmers of particular changes in crop-affecting aphid populations.

The research has only been possible because of careful data management and the evolving systems and software that supported, managed and exploited the data. Notable changes in these systems include:

- In the 1960s the original population data was kept in simple files in a standard format.

- In 1983 the data was put onto a small microcomputer using dBase III.

- Later on, an IBM System/4 was used with a magnetic tape system that gave an editable file system with the raw data. Fortran programmes were written to access, analysis and report.

- In the early 90s the system moved to a 1032 database package running under VAX VMS. This was because the microcomputers became too limited and had 'been under notice of death for six or seven years'. The other benefit of this system was that it got all the data (moth, aphid, and three other species of insect) together.

- In the late 1990s EU funding was used to combine UK trap data with data imported from EU states. The primary objective was to 'integrate existing observing systems at 73 sites in 19 countries to provide a standardised, long term, consolidated, Europe-wide database on aphid incidence'. A Microsoft SQL server and web interface was implemented to allow self-import of data. This system upgrade also permitted substantial analysis and reporting facilities.

Latest developments

Since the EU project, data collection has continued. A known problem has been that over the years 'people have bent the data format' to suit new species and uses. The data curators have seen people 'massaging the data without looking at the underlying software'. As a response to this, in 2009 a new project was initiated to develop a Java application from scratch and to undertake a big data cleanup effort in the process.

[44] Personal communication with Paul Verrier at Rothamsted Research.

Another benefit is that the research team wanted to include other kinds of insect, and so a generalised data format and processing functionality is needed.

After 40 weeks of software development effort - and over 3000 lines of code - the project is at the point of being able to load and use data.

The in-house development approach is to have one software engineer per project so that they can see the project through from beginning to end.

This approach lets the engineer become the expert and minimises miscommunication. Although there is a recognised risk that when people move on important knowledge is lost, there is also the belief that the true understanding of the data is manifested in the scientists, rather than the software engineers - thus mitigating the risk.

The main software packages concern the management of data, since there is a lot of metadata (type of trap, location, *etc*) that needs to be recorded properly, and a 'huge number' of validation rules. The system is still using Microsoft SQL on a single server, though this will be migrated in time to an in-house MySQL server farm. Significant testing is planned including a comparison between the old system and the new systems to see if any data has been left behind or if the basic analysis changes.

Conclusion

The system and software evolution outlined above has maintained access to the data, and provided new functionality and broadened the system's scope. This has allowed new and better research to take place. In practice there have been few 'headaches' around the software, and more around the aging VAX hardware and around the data management. The ongoing software issues are technical problems such as how to get better performance, how to map different data formats to a standardised format, *etc*.

The final sustainability challenge has been that the lead software engineer is now retiring, and due to budget cuts will not be replaced. He is working after his official retirement to ensure that the project finishes and access to data maintained.

Lessons from the software engineering and research project

- **Keep it simple**: for example, describe anything complicated in the code, and use classes and encapsulate wherever possible.

- **Keep track of issues**: this can be simple as the use of a notebook here demonstrated!

- **Generalise from day one**: some of the data management issues arose from […] originally handling different species as different data. Similarly, the data format for a long while was reliant on 80 columns of data as per the original punchcards.

- **Consider 'missing values'**: the original system didn't record zero counts when the traps were operating, so in retrospect it was impossible to tell whether nothing was found that day, or whether the traps weren't used. This has now been addressed by including additional data, for instance what people were looking for, and what they weren't looking for. Zero filling the data table is still infeasible as the data and compute requirements would be too large.

- **Keep your records**: the team had kept the original handwritten records from the traps. This meant that the data curators could go back and look up particular results.

- **Design for reuse**: some of the underlying classes have been used in a different project for curating plant / pathology interactions. This has saved time and money.

Institutionalising education software led to sustainability, and reuse elsewhere[45]

Talks.cam[46] was designed as a clearing house of user-generated event information, to help academics easily publicise seminars they organise, and to learn about intellectually stimulating events in Cambridge which they might be interested in. Organisers of lectures or seminar series can submit details of their events, which are put online on a public website, and which are shared via various methods including RSS feeds, Calendar feeds, email reminders, embeddable website widgets, *etc.*

Talks.cam broke the traditional university software development model, being a grass-roots development project initially, created by academics to meet a need they themselves perceived. As such, it broke several of the traditional tenets of institutional software design and development; for instance, it was not built by a central IT provider or with thought to an eventual place within institutional systems and was not built following traditional 'large IT system' planning and processes. Instead Talks.cam's creators identified their personal needs and rapidly prototyped a system which met them, deploying it on local computers which they had access to through their departmental computer officers. There was no official support for the system, only that provided by the creators in their 'own time'. Some departments and thematic research initiatives began to use Talks.cam wholeheartedly and provided some modest and fairly informal support to the system in terms of staff time. In the long term, this was found not to be sustainable as Talks.cam grew popular and people began to depend on it, without realising in many cases that it was not a fully official and supported system.

In response a project was undertaken to build sustainability of the software and service. A JISC-funded project (~£50k of external funding) the 'Engaging Responses to Emerging Technologies' (EGRET) project successfully institutionalised Talks.cam, a user-generated content and events syndication system. The project had various strands, including technical work to improve the codebase, but also operational (as the organisation absorbing Talks.cam had not prior experience of the Ruby on Rails system) and governance (as control and oversight moved more from the creators to the organisation). From its origins as a grassroots software experiment created by academics, but with potential value to the whole institution, Talks.cam now has been successfully and completely absorbed into CARET, an innovation unit within the University which supports teaching, learning and research activity. This is an exemplar project, demonstrating the process of service handover, from initial idea to full institutional service.

Development, preservation or sustainment?

Though the opportunity was taken to improve the codebase at the same time, it was the need to sustain the service that drove the project. Because only the core functionality needed to be migrated, it could be said that this was preserved.

Lessons

One of the major realisations of the project team (who are now looking to institutionalise other services) is that a robust process is required to ensure that universities are supporting the best applications only, and not committing resources to those of limited utility or high running cost. Useful categorisations for them are software tools that (a) with work may make it; (b) which may look suitable but which settle at a lower stage of growth; and, (c) which although their authors are passionate about them, may never achieve the requirements needed for institutionalisation.

Software reuse

Furthermore, because Talks.cam source code continues to be publicly available under an open source licence it has seen uptake elsewhere, including at the University of Birmingham and at Imperial College.

45 Taken and adapted from *JOS Work Programme: Second Evaluation Report*, Curtis+Cartwright Consulting, V1.0, 17 May 2010 and *EGRET Final Report*, Laura James, V1, 20 August 2009.

46 <http://talks.cam.ac.uk> accessed 4 October 2010.

PARSE.Insight[47] explores preservation for research

"Researchers consider the possibility of re-analysis of existing data as the most important driver for preservation of research data (91%), closely followed by future validation purposes (90%), the advancement of science (89%), and public funding (87%).

At the same time, researchers (as a whole) regarded 'lack of sustainable hardware, software or support of computer environment may make the information inaccessible' as the most important threat to preservation out of the seven threats presented (though this varied by discipline). This is the same as for data managers."[48]

"When asked why data become unusable, all [data manager respondents] reported experiencing situations in which data was lost because the software to interpret the data was no longer available. The lack of contextual information (*eg* manuals, notes, documentation) was also a problem."[49]

The High Energy Physics (HEP) research community responds

"In HEP, preservation of the data is usually pursued only a few years beyond the end of data taking, allowing the analysis to be completed. In most cases the experiment's heritage thus disappears soon after due to the rapid changes in storage technology, computer hardware and software systems.

For any experiment, the potential use-cases define the level of complexity at which the data shall be preserved. The Study Group for Data Preservation in High Energy Physics (DPHEP) came up with different […] levels of increasing complexity (higher-level models, below, include the use-cases of all the lower-level models):

- **Level 1: Provide additional documentation**. An enhanced [level of] documentation would be a recommendation to any preservation model. This could include more information associated to publications, analysis code, detailed information on systematic errors, internal reports, minutes, slides, *etc.* ...
- **Level 2: Preserve the data in a simplified format**. A simple [data] format could be useful for education and outreach purposes but would not be sufficient to perform a full re-analysis. In terms of person-power, this option would require a dedicated expert effort to define the relevant information set for a relatively modest technical implementation and long-term maintenance.
- **Level 3: Preserve the analysis level software and data format**. This level introduces a supplementary dependence on the longevity of the experiment-specific software. More person-power is thus required, and issues of backward compatibility (within the lifetime of the experiment) arise. However, this option would be sufficient for a complete analysis, provided that the existing data is sufficient for the pursued goal.
- **Level 4: Preserve the reconstruction and simulation software and basic level data**. High-level analyses may require "raw", detector-level data and the full simulation and reconstruction software. Significant person-power is needed to prepare for preservation at this level of complexity and to maintain long-term access and usability of the data. However, the clear benefit is the enabling of a full-fledged re-analysis and combination with new data, thus maintaining the full physics potential.

47 PARSE.Insight was a two year European project that started in March 2008.

48 <http://www.parse-insight.eu/downloads/PARSE-Insight_D3-6_InsightReport.pdf> accessed 11 October 2010.

49 <http://www.parse-insight.eu/downloads/PARSE-Insight_D3-3_CaseStudiesReport.pdf> accessed 11 October 2010.

The Study Group recommends that the preservation effort should be built into the experiment strategy at an early stage, such that the archival phase is done with a reduced effort. Open software should be used as much as possible. The data preservation project should start early in order to benefit from the expertise during the lifetime of the collaboration. In the longer term, it should be taken as a permanent activity, implemented in the host laboratory or computing centre associated to the experiment."[50]

Decades of Software Sustainability at the Infrared Processing and Analysis Center[51]

The Infrared Processing and Analysis Center (IPAC)[52] was established in 1986 and originally provided expertise and support for the processing and analysis of data from the Infrared Astronomical Satellite. IPAC was designed to be a center staffed by a team of scientists, development specialists, and administrative staff that would work together closely, and to facilitate frequent interactions with both CalTech and the astronomical community at large.

This concentration of scientific and technical expertise led to IPAC becoming the U.S. science support center for the European Infrared Space Observatory, Wide-field Infrared Explorer, Midcourse Space Experiment, and the Two-Micron All-Sky Survey. As these missions came to an end, IPAC continued its archiving activities and institutional support of the US-based community.

This means that IPAC operates as a long term archive for data from a number of infrared and sub-millimeter astronomy programs handling both the curation of the data, and its dissemination to the community. As a result IPAC has had to take an active approach to software sustainability that seeks to continuously develop the software tools used by the astronomy community, whilst retaining support for all the data archives.

Build an engaged user community that is encouraged to contribute

IPAC embarked on a concerted program of user engagement to attract new users and build a user community. This included user surveys, an end user group (drawn from the community), exhibits and demos at conferences and workshops, advertisements in newsletters, and even "coffee pot" conversations.

Contributions from the user community were not normally code contributions. Instead, the IPAC team actively sought feedback *eg* watching users as they tried services and seeing where they got stuck, and undertaking user surveys where respondents were asked to write down their views, rather than answer questions. This is a more time-intensive approach to start with, but more successful in the long-term: the number of IPAC end-users has increased to 18,000 and 12% of peer-reviewed papers in the area cited IPAC archives or data. This is because IPAC have listened to their users – particularly the advice they didn't want to hear! – and undertaken an active migration approach to software preservation and sustainability.

Legacy software can have a legacy: Modernisation of Scanpi[53]

Written in 1983, the Scanpi analysis code developed and supported by IPAC co-adds scans from the previous far-infrared IRAS survey. It gives a much improved sensitivity gain over survey data products and improves spatial resolution of extended or confused sources.

50 <http://www.parse-insight.eu/downloads/PARSE-Insight_D3-3_CaseStudiesReport.pdf> accessed 11 October 2010.

51 Synthesised from <http://astrocompute.wordpress.com/2010/09/23/software-sustainability-workshop-stories-and-strategies/>, <http://astrocompute.wordpress.com/2010/06/12/one-model-for-software-sustainability/> and <http://astrocompute.files.wordpress.com/2010/09/cardiff_slides.ppt> accessed 3 November 2010.

52 <http://ipac.caltech.edu> accessed 3 November 2010.

53 <http://astrocompute.wordpress.com/2010/06/20/a-case-study-in-software-modernization/> accessed 3 November 2010.

However, Scanpi is also a classic legacy code: written in Fortran 66, it had evolved into a patchwork of scripts and bug fixes, and was a maintenance nightmare. Several dependent modules (eg for data compression) were no longer supported, the software was stranded on Solaris 2.8 and the developer was retiring.

In most cases, it might be assumed that the right approach for Scanpi would be to retire it gracefully, but the IPAC user panel strongly recommended modernisation because of its value in supporting interpretation of data from the current Spitzer and Herschel IR missions.

The software was rewritten from the ground up in C and developed as a workflow application which gave visibility to the processing steps. Several existing components were isolated and refactored for better reuse, leading to a reduction in the code base from 102,000 lines of code to just 21,000 again making the software easier to sustain.

This has led to a legacy application finding a new use, and also represents a compelling argument for user-led migration of software: the existing software took 0.5 FTE to support but by investing a slightly larger effort of 1.25 FTE over a year that maintenance cost has been reduced to 0.1 FTE.

Size and usage of the IPAC archives have grown, but the software and effort to support it has scaled because it's based on common hardware and software architecture, driven by user input.

Lessons in good practice in software sustainability:

- Design for sustainability, extensibility, re-use and portability;
- Build an engaged user community that encourages users to contribute to sustainability; in particular listen to the advice you don't want to hear;
- Be careful about new technologies: do a cost benefit analysis before adopting them;
- Use rigorous software engineering practices to ensure well-organized and well-documented code; in particular control your and manage your interfaces and make source code and test and validation data available.

Collaborative Computational Projects: in it together[54]

The Collaborative Computational Projects (CCPs) bring together leading UK expertise in key fields of computational research to tackle large-scale scientific software development, maintenance and distribution projects. Each project represents many years of intellectual and financial investment.

The aim of the CCPs is to capitalise on this investment by encouraging widespread and long term use of the software, and by fostering new initiatives such as linking to high end computing consortia.

The focus of the CCPs are to provide and support a software infrastructure on which important individual research projects can be built. This means that CCPs are typically grouped around particular research areas, eg electronic structure of molecules, protein crystallography or biomolecular simulation, and they support both the R&D and exploitation phases of computational research projects. CCPs support a relatively small set of researchers (~1000 in total) but represent the majority of the leading groups in each area, which leads to over 500 papers a year (12% in high-impact publications) based on CCP software. This high impact in turn leads to improved chances of sustaining the software.

The main activities of the CCPs are to:

- Carry out flagship code development projects;

54 <http://www.ccp.ac.uk> accessed 3 November 2010.

- Maintain and distribute code libraries;
- Organise training in the use of codes;
- Hold meetings and workshops for users of the codes;
- Invite overseas researchers to engage in collaboration;
- Issue regular newsletters.

A good example is CASTEP.[55] This is a software package which uses density functional theory to provide a good atomic-level description of all manner of materials and molecules. In 1999 a group of UK academics came together to develop a new plane-wave density functional code. This group was to become the CASTEP Development Group. The aims of the project were to produce a clean, stable, portable code which would be easy to maintain and develop. In particular the code was to be designed and specified in advance, with a consistent design philosophy throughout. Since then many new features have been added and continues to be developed actively. CASTEP is marketed commercially by Accelrys, along with Materials Studio, their graphical frontend for Microsoft Windows. In the UK there is an academic distribution, which is available from CCPForge.

The flagship and library model is an interesting approach to ongoing software preservation. Flagship projects represent innovative software developments at the leading edge of a CCP's area of science or engineering. They normally last for three years, funded by research councils, and may support a researcher associated with the project. At the end of a flagship project, the resulting software usually becomes part of the code library supported by the CCP. In some cases, code is released under closed-source licenses which allow income to generated to support ~ 1FTE of staffing specifically for software maintenance.

This flagship model suits most CCPs. It provides a mechanism for responding to advances in the appropriate subject area and maintains the interest of participating staff in cutting-edge research. Other CCPs, especially those involved closely with experimental research (CCP4, CCP14, CCPN), focus more on the collation, standardisation, and development of data analysis codes. Here, it is vital to keep pace with rapid developments in instrumentation.

The CCPs are funded competitively through regular Research Council grants. They have also benefited from support by staff at STFC's Daresbury Laboratory, funded via an agreement with the Research Councils. Such staff provide expert technical and administrative support, including the CCPForge development infrastructure, and are frequently involved in large-scale program development projects. This centralisation of some technical and administrative functions enables more efficient support for the software, but also ensures that best practice can be passed between individual CCPs.

This collaborative approach makes the community able to adapt and respond to developments in computer science, information technology and hardware. One of the strengths of the model is that the focus of each CCP has evolved to maintain international scientific (not just technical) topicality and leadership within its community, whilst supporting the continued use and development of software for over three decades.

Further information and useful resources

Sustainable economics for a digital planet: Ensuring long term access to digital information
http://brtf.sdsc.edu

55 <http://www.castep.org> accessed 3 November 2010.

3.6 Software developer benefits

3.6.1 The benefits given in the sub-sections above accrue mostly to the users of the software, or their organisation, or the higher education sector, or (most broadly) society itself.[56] In order to redress the balance somewhat to the developers who are likely to be contributing to software preservation efforts, some personal level benefits are given below. These, necessarily, are subjective and apply selectively in each situation.

- skill mastery at making longer-lived, more durable software;
- kudos from continued (and wider) use;
- recognition of creativity and problem solving skills (the 'technical challenge of software preservation');
- personal satisfaction from full lifecycle engineering;
- nostalgia value in revisiting old software;
- time and effort savings from better software engineering (that could be used on more interesting problems);
- personal satisfaction from seeing organisational, sector-level or societal benefits ('contributing to the greater good');
- personal satisfaction in formally finishing a task ('achieving closure');
- professional satisfaction in forging new partnerships (*eg* working with archivists or members of an open source community).

[56] Misalignment of incentives was a key theme in the Blue Ribbon Task Force Final Report: "Misalignment of incentives among stakeholders may occur between communities that benefit from preservation (and therefore have an incentive to preserve), and those that are in a position to preserve (because they own or control the resource) but lack incentives to do so." We believe there is a risk of this with software preservation, and so present some benefits to the developer.

This page is intentionally blank

4 Making better decisions about software preservation

4.1 Introduction

4.1.1 This section sets out guidance to help you decide if your software needs to be preserved, and how to go about preserving it. We do not believe that there is a simple and universally applicable formula for determining this, so instead present thought-provoking questions and a range of factors which should be taken into account. This section is best used by reading through and giving careful consideration to those aspects relevant to the software you are interested in.

4.1.2 Note that if at all possible, it's advisable to turn a software preservation problem into a data preservation problem. These problems are invariably easier to handle.

4.1.3 This section is divided into the following three sub-sections that seek to guide on the question posed:

- Does this software need preserving?
- How should this software be preserved?
- Should preservation measures be built into your software development processes?

4.1.4 Considering these three questions should help you decide your strategy and immediate actions for preservation. The first two are interrelated questions, not least because examples have been found where things have been preserved because it was relatively easy to do so[57], and so both should be answered in turn. The first two questions should also answer the question of whether it is viable for the software to be preserved in its current state. In some cases this will be yes, in other cases further development work, further documentation or other activities will be required.

4.1.5 The second of the questions refers to different approaches to software preservation. Our set builds and elaborates on the traditional three used within digital preservation. The set of approaches (and hence options to choose from) are described in Annex A.

4.2 Does this software need preserving?

4.2.1 This question is best answered by considering a set of sub-questions. If clear, positive and compelling answers are available for each, then there is a good case for preservation.

Is the software covered by a preservation policy / strategy?

4.2.2 Firstly, every organisation should have a preservation policy or strategy that covers software. Check whether you have an institutional duty to preserve the software. Where there are external funders involved, there may be small print in your funding agreement.

[57] If the cost of preservation is low and the benefits uncertain but potentially significant then it makes sense to preserve.

4.2.3 When an organisation's preservation policy or strategy is first developed and applied, it may require an audit of all used and held software. The Component Obsolescence Group (COG) details a risk management-based process for undertaking such an audit in an operational computing environment. It is applied by reviewing each system in turn.[58] The process diagram is repeated for illustrative purposes, with kind permission, below:

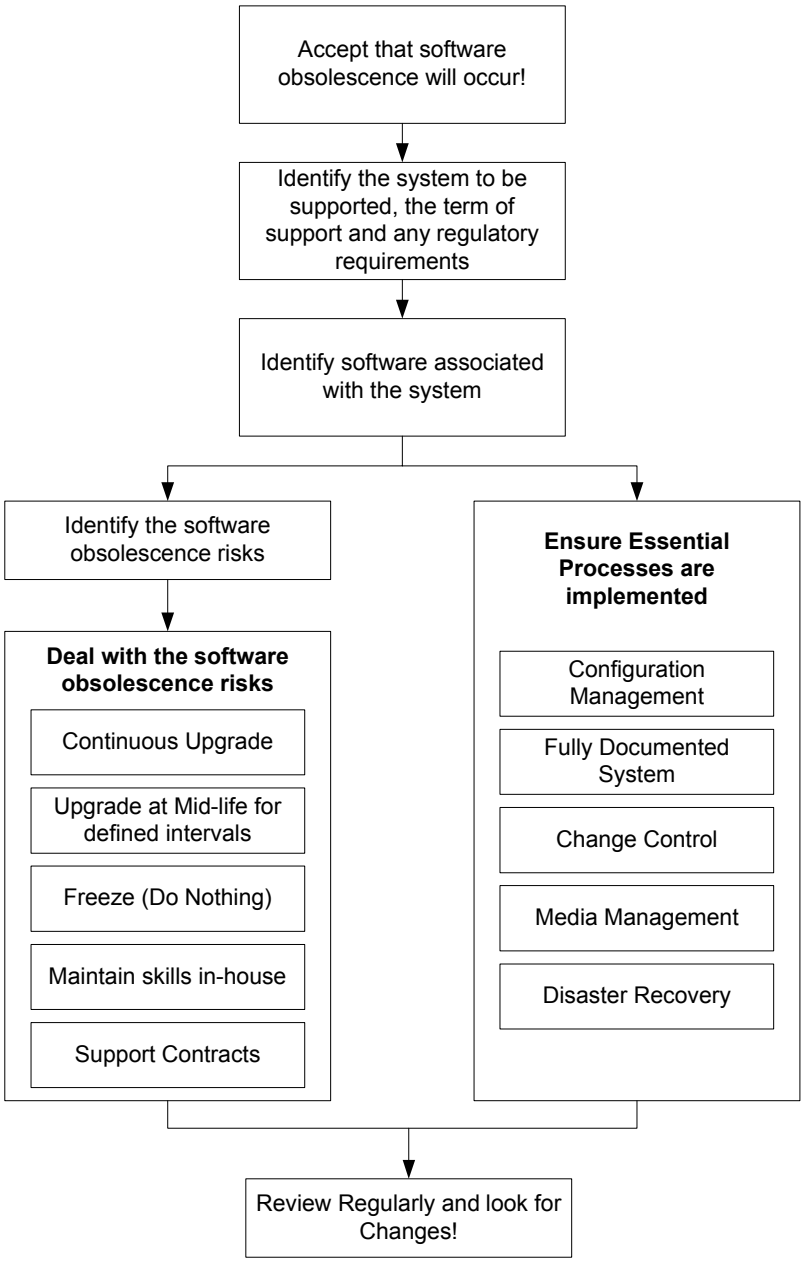

Figure 4-1: COG Process Overview

58 *The Software Obsolescence Minefield*, COG and Graeme Rumney, Issue 1, 2007.

Is there a clear purpose in preserving the software?

4.2.4 Whereas an organisational preservation policy or strategy will have been written with organisational purposes in mind, there may be one or more purposes specific to your situation that then demands preservation. Reviewing the four purposes from the previous section may identify one as particularly relevant. If one can be identified, how compelling is it? A clear and compelling purpose is an absolute must for choosing to invest in software preservation.[59]

4.2.5 With respect to the four purposes from the previous section, some additional pointers are:

- **Encourage software reuse**: The key point here is if the software is suitably mature for reuse. The NASA Reuse Readiness Levels (RRLs)[60] can be used to assess if something is mature enough to release for reuse. If the RRL is 3 or above then reuse is a consideration. If the RRL is 1 or 2, and even at 3 and 4, then further work is required to make it ready for reuse.

- **Achieve legal compliance and accountability**: It should be self evident if there is a mandatory requirement for preservation. What type and degree of accountability are necessary should already be a consideration in all activities, especially publicly funded ones. Working with valuable or sensitive data (*eg* corporate data, medical research, projects relevant to government policy) may fall into this category.

- **Create heritage value**: Based on work by Sellam Ismail at the Computer History Museum[61] some selection criteria for software might be:
  - could reasonably be considered an important example of its type;
  - introduced a new paradigm, product family or launched a new industry;
  - developed using a new and significant software development methodology;
  - serves to demonstrate a significant and colossal failure;
  - significant copies sold or large install base;
  - underlying code has qualities of merit worth preserving;
  - was utilised in something of historical or cultural importance;
  - sufficiently antiquated;
  - supports other preserved software.

- **Enabled continued access to data and services**: This purpose explicitly shifts the focus to the data and/or services that the software supports, and the nature of the relationship between the data/services and software. Software preservation might be essential when data and software can't be decoupled, and when there may be a future requirement to verify results. Software preservation may be desirable when considerable investment has been put into software, and when there is a strong need or motivation to reuse data, for instance where data was expensive to obtain or is non-reproducible.

---

59 "Articulate a compelling value proposition", in the words of the Blue Ribbon Task Force report <http://brtf.sdsc.edu> accessed 7 October 2010.

60 < http://www.esdswg.com/softwarereuse/Resources/rrls/> accessed 14 October 2010.

61 A list of the Software Selection Criteria developed by Sellam Ismail was included as part of a May/June 2005 interview published at <https://softwaretechnews.thedacs.com/stn_view.php?stn_id=1&article_id=30> accessed 30 September 2010. It was noted that the prospective software "must meet one or more, preferably two, of the […] conditions".

Is there a clear time period for preservation?

4.2.6 Ideally as part of a requirement for preservation (rather than a 'nice to have') there would be an associated time period in which the software is stored, retrievable, reconstructable and executable. For instance, if software forms part of a particular records or audit system then there should be a defined retention period. Some Research Councils have mandated certain periods of preservation for research outputs (including software).

4.2.7 Note that the answer does not need to be 'long'. Short or mid-length decisions are fine. As the Blue Ribbon Task Force report says, "a decision to preserve now need not be thought of as a permanent or open-ended commitment of resources over time. In cases where future value is uncertain, choosing to preserve assets at low levels of curation can postpone ultimate decisions about long-term retention and quality of curation until such time as value and use become apparent."

Do the predicted benefit(s) exceed the predicted cost(s)?

4.2.8 Despite the inherent uncertainty in digital preservation issues, some sense of whether the predicted benefit(s) exceed the predicted cost(s) is useful for informing whether some software should be preserved. Benefits were outlined in the previous section, and some of the costs principles associated with particular approaches are set out in Annex A. This report has tried to outline many factors and, as stated earlier, each instantiation of a cost-benefit analysis, or business case, or benefits realisation plan, should, obviously, carefully consider and justify each benefit they assert.

4.2.9 A good question to ask is: how much use is there, and how many users are there? Is this different from anticipated use or users in the future? Usually 'classic' preservation (rather than active development, maintenance or sustainability) is a consideration where the number of users is zero, low or on the decline. Active preservation is more applicable where the number of users is increasing, or could increase, and so there could be immediate benefits from use as well as preservation benefits.

4.2.10 It is not just a question of quantity of use and users, since the utility, impact, dependence, *etc* are all indicators of the value of use. For instance, having many users of a software package that is useful and not critical, and where many alternatives exist, is a different situation from where there are fewer users but where the software offers a unique function and that function is critical to the users' work.

4.2.11 The Blue Ribbon Task Force final report also usefully points out that "The value proposition is not a one-time declaration. Benefits can decline or be eclipsed by other priorities, and the value proposition must be revisited and re-articulated over the course of the digital asset lifecycle. But in all cases, the ultimate threat to persistent access to digital assets occurs when those responsible for preserving the materials decide that the cost of preservation exceeds the perceived benefits to them of long-term access."

4.2.12 The principal disbenefits of allocating effort to software preservation are the costs involved.[62,63] Each approach (see Appendix A) requires some, if not considerable, effort and

---

62 The ITT for this study called for inclusion of 'disbenefits'. These can be thought of as disadvantages. The Office of Government Commerce defines a 'disbenefit' as 'an outcome perceived as negative by one or more stakeholders; disbenefits are actual consequences of an activity, whereas a risk has some uncertainty about whether it will materialise'.

63 As the Blue Ribbon Task Force final report states "In some cases, the benefits of preservation may be most compellingly expressed in terms of negative benefits—the costs incurred if data are not preserved. These costs may reflect the time and effort needed to recreate the information or, if it cannot be recreated, the kinds of uses that would then not be possible. For classes of data that carry ethical issues—human subject and animal research,

perhaps financial investment too. Also important is the opportunity cost – *ie* if not used for preservation, what outcomes could have been achieved with the best alternative use of the effort and money? For instance, if a researcher who has coded some software for their academic use and then polished the code and archived it on sourgeforge.net but then it sees no further use – could that time have been better spent conducting research, formulating new ideas, attending a conference, *etc*? The potential loss of not preserving software (*ie* all the activity that it could have enabled) also bears consideration. All costs are important given the often inherent uncertainty about the long-term value from software preservation.

Is there motivation for preserving the software?

4.2.13 With digital preservation there is often a misalignment of incentives. And, even if the person with responsibility, power or expertise has the incentives they still may not be interested in using their time for software preservation. Finding someone with drive and enthusiasm would help.

Is the necessary capability available?

4.2.14 Certain technical capability (systems, skills, *etc*) and non-technical capability (information, soft skills, support, *etc*) might be required for a particular preservation approach. Understanding whether the right capability is available is important.

archaeological sites, or extinct species and languages—the benefits of preservation are often better framed as mitigating the risks of unacceptable loss—unacceptable because the loss violates shared ethical standards."

4.2.15 A good example of this is the decision flowchart from the SigSoft team. This highlights the importance of whether the significant properties are available and elicitable (via personnel, documentation, *etc*). This flowchart is repeated below:

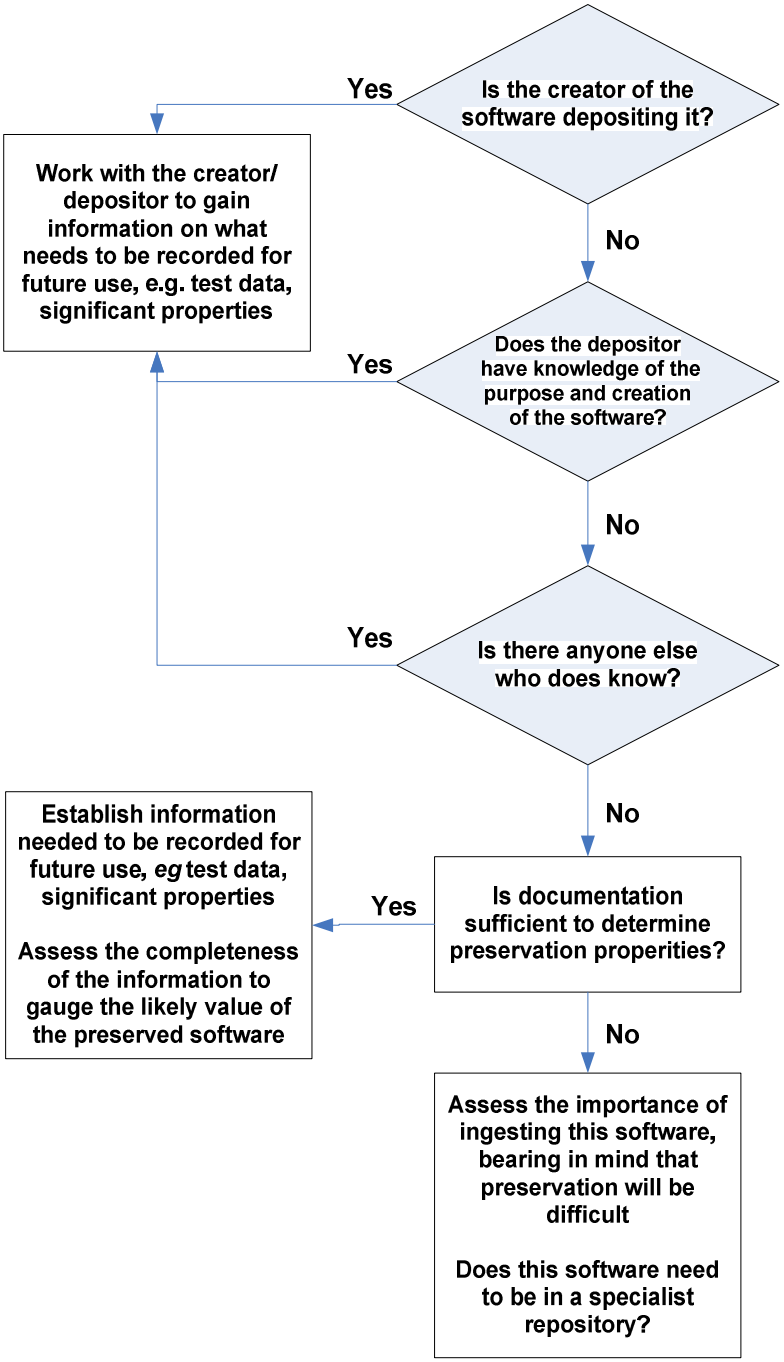

Figure 4-2: SigSoft Process Overview

4.2.16 Project management capability (*eg* planning and risk management skills) may well be required, especially if the proposed preservation activities are substantial.

Is the necessary capacity available?

4.2.17 This question addresses whether preservation is affordable, both in terms of effort and money. You could consider different models, for example bringing in specialists if the necessary capacity is not available in-house.

4.2.18 It should be noted that availability and affordability of capacity changes over time, another justification for viewing preservation activities as an ongoing review process, not a one-off decision.

4.3 How should this software be preserved?

4.3.1 So, you have decided to preserve your software. How should you go about it? Seven different approaches to preservation are detailed in Annex A, and are summarised here as:

- **Technical preservation (techno-centric)** - Preserve original hardware and software in same state;
- **Emulation (data-centric)** - Emulate original hardware / operating environment, keeping software in same state;
- **Migration (functionality-centric)** - Update software as required to maintain same functionality, porting/transferring before platform obsolescence;
- **Cultivation (process-centric)** - Keep software 'alive' by moving to a more open development model, bringing on board additional contributors and spreading knowledge of process;
- **Hibernation (knowledge-centric)** - Preserve the knowledge of how to resuscitate/recreate the exact functionality of the software at a later date;
- **Deprecation** - Formally retire the software without leaving the option of resuscitation/recreation;
- **Procrastination** - Do nothing.

4.3.2 A whole range of considerations are pertinent to choosing between these options. These include:

- How much access do you have?
  - Are you the owner of the code?
  - Are you the developer of the code?
  - Do you have access to the source code?
  - Do you have access to the hardware the software is running on?
  - Are you a user of the software?[64]
- Do you have the necessary Intellectual Property Rights (IPR)?
- What are you needing to preserve?

64 Even end users can influence whether software is sustained. For example, weight of customer opinion forced Microsoft to extend its support for Windows XP when it perhaps would have preferred wholescale migration to Windows Vista. On the smaller scale of niche academic software user opinion should carry greater weight.

    - A few major pieces of functionality;
    - Most of the functionality, but tolerant of minor deviations;
    - All functionality, but fixing errors when found;
    - Must perform exactly as original.

- What is your likely effort profile?
    - Something now, nothing in future;
    - Something now, something in future;
    - Nothing now, something in future;
    - Nothing now, nothing in future.

- What is the maintainability of underlying hardware?

- Is maintaining integrity and/or authenticity an important requirement?[65]

- How long do you want to preserve it for?
    - In general, source code can be preserved for longer than binaries.
    - Design documents, pseudo-code, test data, *etc* lasts longer than source code.
    - Standards last longer than software product series, which last longer than software versions.
    - If software needs to be preserved for a decade or more, this is multiple generations of technology.

- Can you afford it?
    - Do you have the necessary funding and effort to commit to preservation?

- Are you also interested in further development or maintenance?
    - For example, new or improved functionality might be a parallel interest.

- What development effort has been invested into the software so far?
    - Whilst 'sunk costs' should not influence whether software is preserved, the effort that has gone into the software may reveal whether code or binaries should be preserved, or should efforts go into preserving the design, test data, algorithm, *etc*. For instance, if a trivial script was written to produce a particular graph for a paper and the method has already been documented in the algorithm, then perhaps it is better to retain the capability to rewrite the software rather than preserve the software with other approaches.

- Is the software already open source, or could it be made open source?
    - Are there any barriers to making it open source?

[65] Sometimes maintaining the chain of custody is critical. For example, the preservation of some computer games has required that the 'taking off of the shrink-wrap' is logged and tracked.

Is each approach appropriate to every purpose?

4.3.3 No. Some approaches are better suited than others to each purpose. The table below provides an indicative mapping between the four purposes and appropriate approaches.

	Technical preservation	Emulation	Migration	Cultivation	Hibernation
Achieve legal compliance and accountability	✓	✓	✓		
Create heritage value	✓	✓			
Enable continued access to data and services	✓	✓	✓	✓	✓
Encourage software reuse			✓	✓	✓

4.3.4 We do not consider procrastination to be an appropriate approach to any software.

What are the relative advantages and disadvantages of each approach?

4.3.5 Understanding the advantages and disadvantages of your intended approach is key to making a final decision. The table below sets out some advantages and disadvantages.

Approach	Advantages	Disadvantages
Technical preservation (techno-centric)	– Clearly defined approach – Can often change to emulation at later date – Access to source code not necessary – Will perform exactly as original – Minimal technical capability required – Well suited to embedded systems	– Risky to rely on hardware and software which is no longer supported or is considered obsolete – Does not guarantee future access if dependent on other hardware/software (*eg* networking) – Can be vulnerable to malicious attack – Access may be limited to a specific physical location – Use is limited to those users with the specific hardware / software setup – Long-term hardware degradation make this a short-term approach

Approach	Advantages	Disadvantages
Emulation (data-centric)	– Little technical capability and effort required if reliable emulator exists – Easier to manage virtualised hardware – If emulator continues to be developed/supported, software can continue to be run indefinitely – Access to source code not necessary	– Need all aspects of hardware to be emulated correctly, including flaws – Should perform exactly as original but there may be subtle nuances or annoying quirks; at worst it ought to provide general functionality but sometimes there are more serious problems – May need a licence for emulator – If emulator ceases to be developed need an alternative approach – Can be vulnerable to malicious attack – Performance often suffers
Migration (functionality-centric)	– Allows further continued maintenance of software – Allows further development of software for new or enhanced functionality – Enables access on other platforms – Source code approaches likely to outlast binary based approaches	– Requires intellectual property of software to be held – Requires continued effort for maintenance and porting – Requires enduring technical capability – Little guarantee that it will perform exactly as original, but major functionality should be preserved and is better for where there is tolerance of minor deviations – Frequency of update is unpredictable (technologies are volatile)
Cultivation (process-centric)	– Shares the sustainability workload – Increases chances of continued maintenance of software – Allows further development of software for new or enhanced functionality – Potential for better migration to other platforms – Source code approaches likely to outlast binary based approaches	– Not a quick nor guaranteed fix (building a self-sustaining community takes time and often fails) – Requires intellectual property of software to be held – Requires more coordination – Possibility of loss of control of direction – Technical and community-building capability required
Hibernation (knowledge-centric)	– Useful when you have a known break in effort – Supports repeatability/reproducibility of results as the same or similar algorithm can be re-implemented – Documentation-based approach could be longest lived	– Can be difficult to check if hibernation processes are rigorous until after it is too late – Requires familiarity and understanding of software – Returning to any code after a long delay will always be hard
Deprecation	– Threat of deprecation may stir latent users into action – Unambiguous statement of status of the software	– Closes option of future use, which is often hard to predict – May require software archaeology skills in the future

Approach	Advantages	Disadvantages
Procrastination	– Comes naturally – No upfront cost – May be the best approach where trivial effort was needed to develop software, or it was a useful training exercise, and there are no other users	– Entirely reactive: not a valid preservation approach! – May require software archaeology skills in the future – Largest later cost, if software is needed again

Identifying purposes, benefits and scenarios for the DMAREL software

The purposes, benefits and approaches in this section have been applied retrospectively to the DMAREL software, showing how a rationale can be created and a preservation strategy (migration with a hint of emulation) chosen. DMAREL is an application in the domain of computational chemistry that allows energy minimisation of rigid molecules with the electrostatics described by a distributed multipole. It was written by Maurice Leslie who was in the Computational Chemistry Group (CCG) at Daresbury Laboratory under the auspices of the Engineering and Physical Sciences Research Council (EPSRC) for the EPSRC's Collaborative Computational Project for the Computer Simulation of Condensed Phases (CCP5).[66]

DMAREL was a key component of a computational workflow used by Prof Sally Price's group at UCL in order to predict the theoretical existence of different crystal structures of molecules.

Motivations

However, there was an increasing need to move on from DMAREL. A key motivation for this was whilst DMAREL could successfully predict crystal structures on the size of *eg* a Benzene molecule, the UCL research team had requirements to predict structures for much larger molecules. Another motivator was that the major author and maintainer of DMAREL was in the process of retiring but was maintaining the software on a goodwill basis which was clearly not sustainable.

As part of the CPOSS (Crystal Prediction of the Organic Solid State) project[67], Prof Price's group decided to develop a new codebase themselves. In order to achieve this, they first negotiated the intellectual property transfer from Daresbury to UCL.

The Migration and Deprecation Process

Following this, they began a process of knowledge transfer from the original author to the present team. Still a Fortran application, they were constrained in that DMAREL formed part of a much wider community of use where it was embedded in other application workflows.

The replacement would have to be backwards compatible with respect to all inputs, outputs and application options to ensure plugin-compatibility into existing DMAREL related workflows.

66 <http://www.cse.scitech.ac.uk/ccg/software/DMAREL/index.shtml> accessed 14 October 2010.

67 See DMACRYS pages on <http://www.cposs.org.uk> accessed 14 October 2010.

This essentially meant they had to work to understand the current usage of DMAREL outside of their own group in other workflow ecosystems. Having done this, they replaced DMAREL in their own workflow with DMACRYS to validate its behaviour, which was successfully achieved within the ENGAGE-CPOSS project[68]. They clearly recognised the need to duplicate previous results from DMAREL with DMACRYS, and went to great lengths to verify that workflows where DMAREL was used produced identical results when replaced by DMACRYS.

With the new version, not only could UCL do bigger scale science, but anyone using DMAREL could use DMACRYS in the same environments, which acts the same as DMAREL in all appropriate respects. Now they are confident DMACRYS is a suitable successor to DMAREL, they have taken ownership of links with DMAREL and are actively deprecating it. This requires actively seeking changes to the website of the original supplier and replying to any requests for DMAREL with e-mail that strongly encourages that new users use DMACRYS and warning that DMAREL is no longer supported.

The principle preservation approach was migration of the software's function to new software/There is also an element of emulation - in the sense of 'pluggable' compatibility within an application environment (*ie* precise emulation of previous input and output formats, options and application behaviour). This extends the emulation concept beyond the emulation of an operating system or other similar environment. Finally the original code was actively deprecated in order to prevent bifurcation of the user community.

How this relates to the framework

Encourage software reuse

The relevant scenarios within this purpose in the framework are:

- Continuing operational use in institution;
- Increasing uptake elsewhere.

But in addition, they needed a superior product with the constraint that it was backwardly compatible with its predecessor.

The benefits were:

- Increased quality and dependability – the codebase was reengineered and provided improved and larger-scale functionality;
- Focused use of specialists – by centralising development within UCL;
- Reduced duplication - superseded and actively deprecated predecessor;
- Opportunities for commercialisation - pharmaceutical companies would find it impossible to be able to analyse modern drug candidate compounds using DMAREL - too small-scale.

Enable continued access to data and services

The relevant scenarios within this purpose in the framework are:

- Reproducing and verifying research results – the emulation aspect;
- Repeating and verifying research results (using the same or similar setup);

68 <http://www.engage.ac.uk/documents/engage/pb-summary-pdfs/ENGAGE%20Project%20Brief%20Crystal%20Energy%20Landscape%20-%20public.pdf> accessed 14 October 2010.

- Reusing data in combination with future data – can re-analyse previous data in more detail and in more ways;
- 'Squeezing' additional value from data – more analysis detail can be provided from prior input data;
- Identifying new use cases from new questions;
- Maintaining legacy systems (including hardware) – the CPOSS ENGAGE project was not possible without development of DMACRYS; DMAREL needed to keep operating for a while so that they could validate results;
- Ensuring business continuity;
- Avoiding software obsolescence – the most important in this case.

The benefits in terms of data were:

- New insight and knowledge – due to the magnitude of the science now possible with DMACRYS;
- Increased assurance in results.

Additionally, it enabled them to raise their profile within field. And although not a primary goal, a 'nice-to-have' was the commercial licensing opportunities that grew out of this work since DMACRYS could meet the scale required by industry.

The benefits in terms of services were:

- Current operations maintained – the revised software is fully backwards compatible with its predecessor;
- Opportunity for improved operations via corrective maintenance – the software is being developed within a more sustainable environment so corrective actions are more tractable.

An additional benefit, in itself, is that the team is more sustainable as a software development group.

Lessons learnt

- The effort required for the transfer of intellectual property was underestimated - observing the legal formalities was time-consuming but strictly necessary in order to clarify the IP position.
- From a development perspective, adoption of a source code Revision Control System is far more sustainable than their previous manual process of merging updates.
- Good technology knowledge transfer is very important in migration of development to the new team; they were fortunate to have the previous software author as a retired consultant.
- Understanding of the wider ecosystem of DMAREL was critical to understanding how best to approach the development of DMACRYS – the identified need to maintain the Input / Output (IO) formats and options was critical.

4.4 Should preservation measures be built into your software development processes?

4.4.1 A principle from other areas of digital preservation[69] and a strong lesson from the case studies in this report is that considering preservation, sustainability, *etc* upfront (and again regularly) is important. This would imply that building preservation measures into software development measures is good practice. But what are these 'preservation measures'? Two measures are apparent:

- **Software engineering**: As sub-section 2.5 stated, 'good software preservation arises from good software engineering' so encouraging better software engineering practice will benefit software preservation if and when required. Software engineering practice includes the Significant Properties concept.
- **Identifying explicit preservation requirements**: Requirements capture and management is an upfront activity in software development, and preservation requirements should be considered along with other requirements on functionality, interfaces, performance levels, security/privacy, *etc*. Explicitly agreeing and building in preservation requirements means that the specification, design, maintenance, *etc* will all reflect the need for preservation. Changing preservation requirements can be accommodated as with any other type of requirement.

4.4.2 The extent to which both software engineer practice and preservation requirements should be a priority depends on both the intended functionality of the software (*ie* whether it fits one or more of the four purposes) and the nature of the software itself. For example, is the software meant to be a proof-of-concept demonstrator, something more heavyweight like a pilot, or perhaps an operational service for a defined set of users? Each allows a different approach to be taken with different expectations of robustness and longevity. For instance, the development of large-scale, complex software always benefits from software engineering, so this supports software preservation too.

4.4.3 A strong analogy can be made to the well-understood place of risk management in software development processes. By building software preservation measures into existing software development processes is similar way to the application of risk management measures through the development of a piece of software. One can therefore take an appropriate approach to preservation dependent on the many factors (software maturity, impact of use, availability of effort *etc*) and review this as part of the ongoing software development process.

---

69 For example, an action articulated in the Blue Ribbon Task Force final report was "Take preservation steps early in the digital lifecycle; create and codify contingency plans; make and implement plans for handoffs to address economic risks over the digital lifecycle.", Sustainable economics for a digital planet: Ensuring long term access to digital information, Final Report of the Blue Ribbon Task Force on Sustainable Digital Preservation and Access, February 2010, <http://brtf.sdsc.edu> accessed 5 October 2010.

A Different approaches to software preservation

A.1 Summary

A.1.1 This annex provides a brief overview of different approaches to software preservation. This helps set the context and guidance in the main body of this framework document. The approaches set out are an extended set from those traditionally covered.[70] This is done to give proper coverage of approaches specific to open source software, to include a more pragmatic option that gives additional flexibility and to include a 'default' option for a baseline. The different approaches therefore covered are:

- **Technical preservation (techno-centric)** - Preserve original hardware and software in same state;
- **Emulation (data-centric)** - Emulate original hardware / operating environment, keeping software in same state;
- **Migration (functionality-centric)** - Update software as required to maintain same functionality, porting/transferring before platform obsolescence;
- **Cultivation (process-centric)** - Keep software 'alive' by moving to a more open development model, bringing on board additional contributors and spreading knowledge of process;
- **Hibernation (knowledge-centric)** - Preserve the knowledge of how to resuscitate/recreate the exact functionality of the software at a later date;
- **Deprecation** - Formally retire the software without leaving the option of resuscitation/recreation;
- **Procrastination** - Do nothing.

A.1.2 Each approach is provided with a description, a set of activities, notes on costs and some pointers to further information and useful resources.

A.1.3 Some metrics (or more general indicators) are also proposed to determine if the approach is going to plan. These will need to be tailored to the specific software preservation plan being used. A suitable set of metrics will inevitably cover a broader scope than software preservation (*eg* the software development process) and include technical, process and economic factors. Some concepts and examples are identified for each approach.

A.1.4 Good questions to ask in undertaking each approach are: What would happen if the lead developer were hit by the proverbial bus? What would happen if the project (or organisation) were to be shut down urgently? Continuity (in each of the four aspects of storage, retrieval, reconstruction and replay) is necessary for software preservation.

A.1.5 This annex can be used either as a refresher for the different approaches or as a starting link to a more detailed investigation. For a comparison of the different approaches see sub-section 4.3.

---

70 The three main preservation approaches to digital preservation – Technical Preservation, Emulation and Migration - and their origin are described in the Cedars Guide to Digital Preservation Strategies (2002), <http://www.webarchive.org.uk/wayback/archive/20050410120000/http://www.leeds.ac.uk/cedars/guideto/dpstrategies/dpstrategies.html> accessed 7 October 2010.

A.2 Technical preservation (techno-centric)

A.2.1 Technical preservation is a planned and intentional decision to keep the software and hardware running in the same state. There is also the option of purchasing and spares so that components can be replaced as they fail. Bear in mind that no obsolete technology can be kept functional indefinitely. Technical preservation generally works best for when there is a known preservation period (especially if this is less than known support periods).

A.2.2 Software is reliant on hardware, and hardware changes as each new model is released. Over time, hardware will change to such an extent that older software will not run on the latest hardware. Without the hardware to run on, software becomes redundant.

A.2.3 The easiest way to ensure that there will always be hardware to run your software is to preserve the hardware. Technical preservation has one big benefit: it's easy. You simply continue business as usual. There are drawbacks to this approach. The first is maintenance. Over time, hardware components will wear out and must be replaced. If the hardware is no longer manufactured, components become scare and expensive. Ultimately, you may find yourself with broken hardware and no way of fixing it – leaving you with redundant software. The second drawback is isolation. Your software only works with very specific hardware, which limits your users to those people with the right hardware. This might be a very small group.

A.2.4 Technical preservation is a straightforward approach to sustainability, but it's only as reliable whilst you have a stockpile of spare parts.

A.2.5 Specific activities within this approach are likely to include:

- purchasing spares;
- regular checking that the system works;
- maintaining hardware;
- replacing hardware elements as they fail;
- scheduling review points in the calendar.

A.2.6 There are some points to factor in about the costs of this approach:

- Some upfront costs to purchase spares;
- Low cost initially (maintenance only) to keep the hardware and software running;
- Costs likely to rise over time as maintenance gradually becomes more difficult;
- At some point a large cost will be incurred as hardware fails and a replacement approach is necessary.

A.2.7 Some metrics or indicators to monitor to know how well this approach is proceeding could be designed around one or more of the following:

- continued executability (*eg* percentage monthly uptime, or number of failures a month);
- ongoing maintenance overheads (eg effort per month, or direct/indirect cost per month);
- number of remaining spares;
- expected cost of a replacement system (NB: this will change non-linearly over time).

Data preservation, not software preservation[71]

The UK Data Archive (UKDA) is curator of the largest collection of digital data in the social sciences and humanities in the UK. Since the 1970s it has taken a data migration-based approach to preservation. This seeks to entirely separate and decouple data from the software. This strategy was founded on the premise that, at some time in the future, the software used to create or analyse data will not be executable. Therefore both software and platform neutrality is critical. The alternate strategy would be to maintain and disseminate (or provide online access to) older versions of software alongside the legacy file formats. While this approach is increasingly technically feasible with virtual machines it still faces significant challenges over the rights to use and disseminate commercial software packages and limits users access to functionality offered by modern revisions of those packages. They still need the software applications for data they are going to curate in order to prepare the data for migration, but placing complete dependence on the existence of new versions of the software would be to avoid the questions of software dependency. Implementing this strategy is made easier because they actually have few software packages to consider: ten packages cover around 95% of the data. They also periodically announce the software packages they can support to retain their focus. However, because they need to migrate data (from old to new) the software they run is sometimes unsupported and legacy. Their approach is strictly to support this software rather than preserve it; and this support will diminish over time, as fewer instances of legacy file formats are deposited for ingest by the Archive. The Archive's Preservation Policy is: <http://www.data-archive.ac.uk/curate/preservation-policy>.

This approach does mean that there are challenges around hardware maintenance and backwards compatibility of different versions of software. An instance of these challenges was faced with SPSS, the common statistical package. The UKDA wrote scripts in 2003 for their workflow. These scripts were dependent on running v13 of SPSS. Because of the significant investment in writing the scripts (approximately one-person year of specialist effort) the UKDA continued to run machines with v13 on them for as long as possible. Eventually the support overhead for the obsolete software and hardware became untenable and the team created a data conversion tool that could convert between SPSS file formats for any version between v6 and v18. This tool allowed them to retain their data migration-based approach to digital preservation. However, this too will need to be upgraded periodically.

The tool was created as part of a 12 month JISC-funded project - Data Exchange Tools and Conversion Utilities (DExT). The final report[72] from the project reaffirmed the importance of open data exchange formats: "Data conversion and proprietary data entry and analysis are particularly important and problematic aspects of data management and curation... The main issues involve the buying-in to a dedicated analytic strategy and typically a particular software package. Over the years the UKDA has seen a number of such softwares quickly become obsolete. To address the problem of incompatibility between software various data conversion tools have come of the market. However in the qualitative data analysis software field there are no such inter-software conversion tools. Open data exchange formats are necessary for maximising the opportunities for data sharing and long-term archiving."

Further information and useful resources

The National Museum of Computing
http://www.tnmoc.org

Computer History Museum
http://www.computerhistory.org

Keeping Old Computers Alive
http://www.techsoup.org/learningcenter/hardware/archives/page9667.cfm

71 Personal communication with Matthew Woollard, 11 June 2010.

72 <http://ie-repository.jisc.ac.uk/393/> accessed 14 October 2010.

A.3 Emulation (data-centric)

A.3.1 You want to keep your software, but you're worried that technical-preservation might leave you with no hardware or an expensive maintenance bill. The alternative may be emulation. An emulator is a software package that mimics your old hardware and/or operating environment, and can be run on any computer.

A.3.2 Emulation gives you the flexibility to run your software on new hardware, which gives your software a new lease of life. As always, there are drawbacks. You need to find an emulator. You might be lucky and find one available under a free-to-use licence, or you might be able to buy one. However, if your old hardware was rare, you may find that no emulator exists. In this case, you either have to write an emulator yourself, which requires specialist skills and could be expensive, or explore another of the sustainability approaches. It is difficult to write an emulator that perfectly mimics the old hardware. This can lead to differences between the operation of the old hardware and the new emulator, which could manifest themselves in annoying quirks or more serious problems.

A.3.3 Specific activities within this approach are likely to include:

- regular checking that the system works;
- regression testing;
- verifying and validating results;
- updating the emulator (or maintaining it if developed in-house);
- scheduling review points in the calendar.

A.3.4 There are some points to factor in about the costs of this approach:

- Low cost if emulator exists as costs borne by someone else;
- Emulators themselves need sustaining;
- At some point a large cost may be incurred as emulator ceases to work and a replacement approach is necessary.

A.3.5 Some metrics or indicators to monitor to know how well this approach is proceeding could be designed around one or more of the following:

- continued executability (*eg* percentage monthly uptime, or number of failures a month);
- frequency of updates to emulator to know if the emulator being sustained (*eg* updates per quarter, number of unresolved bugs);
- cost of emulation (*eg* total costs in licensing, handling emulation errors, verifying data, *etc* per month or quarter);
- emulation performance (*eg* average response time over a month for a service or the program execution time for a command-line application based on some sequence of test queries or input).

Further information and useful resources

PLANETS project
http://www.planets-project.eu/docs/reports/Planets_PA5-D1-TestingToolsForTechnicalEnvironments-Final_v2_public.pdf

Project KEEP
http://www.keep-project.eu

A.4 Migration (functionality-centric)

A.4.1 Migration keeps the system functional with new technology.

A.4.2 If you need to reproduce the operation of your software reliably, the best choice may be migration. With this approach, you re-code your software so that it will work on new hardware or operate with new reliable software. Re-coding for migration also gives you the perfect opportunity to enhance your software's operation, such as fixing bugs or adding new features.

A.4.3 There is a wide range of migration approaches from a complete re-write of the code, which allows the software to be used on a completely different system, to continual migration, which keeps your code up to date with the latest (generally small and continual) changes to the hardware and software that your code relies on.

A.4.4 A complete migration to a new system is the same as writing new software – possibly harder because you are constrained by the old architecture. This leads to the biggest drawback: it's resource heavy. Dependent on the complexity of your old software, you may need a lot of development time to be invested into the migrated code.

A.4.5 At some point in its lifetime, most code will be subject to a change in the hardware and software that it relies on. For example, your code might need to be tweaked to use a new version of Java or the latest version of an operating system.

Technology obsolescence in industry

"Peter Sandborn is a Professor in the CALCE Electronic Products and Systems Center at the University of Maryland. Dr Sandborn's group develops obsolescence forecasting algorithms, performs strategic design refresh planning, and lifetime buy quantity optimization."[73]

Obsolescence is a major issue in industries with long lifetime systems, since constituent components increasingly become obsolete well before the system's intended end-of-life. Peter describes it thus: "To deal with that growing pile of unavailable supplies, engineers in charge of long-lasting systems must basically predict the future--they must learn to plan well in advance, and more carefully than ever before, for the day their equipment will start to fail… **Call it the dark side of Moore's law: poor planning causes companies to spend progressively more to deal with ageing systems**".[74]

The answer he puts forward is refresh planning. "The goal of refresh planning is to find the best date to upgrade a product and to identify the system components on which the redesign should focus". Peter has "developed one such methodology, called Mitigation of Obsolescence Cost Analysis, or MOCA, which determines when a design refresh should occur, what the new design should accomplish, and how to manage the parts that go obsolete before that time... The key to a successful refresh schedule is deciding on it well in advance, so that a project's budget can include that expense before irreplaceable parts become a serious business liability."[74]

73 <http://www.glue.umd.edu/~sandborn/> accessed 4 October 2010.

74 <http://spectrum.ieee.org/computing/hardware/trapped-on-technologys-trailing-edge> accessed 6 October 2010.

Whilst Peter's main interest is in hardware obsolescence, he sees software obsolescence as "a concurrent problem". As his paper[75] on software obsolescence (one of the few on the topic) says "in reality [addressing obsolescence] is a hardware/software co-sustainment problem, not just a hardware sustainment problem" Peter also says "We do handle software obsolescence in MOCA, *ie*, we do lots of refresh planning studies where the bill of materials is composed of a mix of hardware and software. One thing we find is that software often creates constraints on the refresh planning process for the hardware. For example, the end of support date for an operating systems creates a constraint that the operating system has to be off all fielded systems and cannot be used in new systems, which in turn means that there has to be a refresh that includes changes to the operating system and possibly associated hardware prior to the end of support date for the operating system."[76]

A.4.6 The effort required for migration varies widely from small changes (*eg* reconfiguration, made necessary by an evolving platform, and easy recompilation), to major updates every so often (*eg* rewriting the code in a new programming language). At its most extreme, migration (whether part of continuous upgrade or a scheduled mid-life upgrade) involves completely redeveloping software from the original requirements, specification or design.

A.4.7 Development to improve the functionality, user experience, *etc* can be done at the same time as migration, though the primary purpose of migration is to preserve the function and ensure future maintainability.[77] An improvement in performance is often another benefit of migration as the system is now likely to be operating on a more modern, powerful hardware and software platform.

A.4.8 A particular type of migration is that of moving to an open licence and/or a community-based development approach. This is covered explicitly in the next sub-section on Cultivation.

A.4.9 Specific activities within this approach are likely to include:

- reconfiguring and recompiling ('porting');
- learning and using new programming languages;
- (in extremis) rewriting the original code from the specification (*ie* re-engineering the system);
- (in extremis) reverse-engineering from a binary file;
- scheduling review points in the calendar.

A.4.10 There are some points to factor in about the costs of this approach:

- The cost of continual or intermittent migration is likely to match or exceed the initial development cost.

A.4.11 Some metrics or indicators to monitor to know how well this approach is proceeding could be designed around one or more of the following:

- continued executability (*eg* percentage monthly uptime, or number of failures a month);

---

75 *Software Obsolescence – Complicating the Part and Technology Obsolescence Management Problem*, Sandborn, IEEE Trans on Components and Packaging Technologies, Vol. 30, No. 4, pp. 886-888, December 2007. The paper identifies three main causes of COTS obsolescence: functional obsolescence, technological obsolescence (end-of-sale, legally unprocurable or end-of-support) and logistical obsolescence. It should also be noted that, electronic parts obsolescence is a well studied field but "the one common attribute of all the methodologies, databases and tools that are in use today, whether reactive, proactive or strategic, is that they focus exclusively on the hardware life cycle. In most complex systems, software life cycle costs (redesign, re-hosting and re-qualification) contribute as much or more to the total life cycle cost as the hardware, and the hardware and software must be concurrently sustained."

76 Correspondence with Peter Sandborn, May 2010.

77 Though in reality 'perfective maintenance', where the primary purpose is to improve the software for users, is more likely.

- continued compilability (*eg* binary yes/no for compilation failure, or number of compilation warnings[78]);
- cost of maintenance (*eg* developer hours per month);
- coverage of supported system functions (*eg* the ratio of functions that are supported by the migration approach to the total number of functions upon which the system is reliant).

Further information and useful resources

The Software Obsolescence Minefield (Component Obsolescence Group)
http://www.cog.org.uk

DoD Software Migration Planning
http://www.sei.cmu.edu/reports/01tn012.pdf

Porting: A Development Primer
http://www.mindfiresolutions.com/mindfire/Porting_DevelopmentTechniques.pdf

Commercial technology translation
http://www.greatmigrations.com/resources_articles.aspx
http://www.semdesigns.com/Products/Services/LegacyMigration.html

78 For instance it could be telling if the code uses a deprecated API and the compiler flags a warning: it still works now but could fail if support for that API is withdrawn.

A.5 Cultivation (process-centric)

A.5.1 Sustainability requires the investment of resources, and cultivation is one of the best ways of sharing the responsibility for these resources. Cultivation is the process of opening development of your software. This is where you allow developers access to your code – under a licence – so that they can work with you. The deal is that outside developers can develop your software so that it meets their exact needs, and in doing so, any bugs they fix or new functionality they add can be given back to your project.

A.5.2 Cultivation allows more contributors to be brought into your project, which helps share the sustainability workload. With more people, knowledge about your software is spread over a wider group, so that the departure of one person is less likely to affect the software's future.

A.5.3 Cultivation's main drawback is that it's a long-term process. Cultivation is not suitable as a quick fix to ensure sustainability in the short term; instead it requires effort and planning over many months and years. Moving to open development is not as simple as making your source code publicly available. You also need to build a community around the software, and this requires work to understand your community and how to appeal to them. Once in place, your community could become self-sustaining so that the future of your software is assured.

A.5.4 Cultivation promises a self-sustaining community of developers who work together to keep your software up to date, but requires work to cultivate the right community for your software. A combination of Open Source licensing and Open Development practices make it easier to preserve software by removing barriers to others taking on the preservation of the code. The body of knowledge about a piece of software is more likely to be manifested in electronic form, as opposed to being held in the heads of a few developers. However it is important to reiterate that OSS alone is not enough to enable the preservation of software - this also requires aspects of curation and ongoing minimal maintenance to cope with environmental changes.

A.5.5 Specific activities within this approach are likely to include:

- choosing an appropriate open source licence;
- applying an open source licence to an existing codebase;
- moving code to an open source repository (*eg* sourgeforge.net);
- setting up a development website, mailing list, *etc;*
- cleaning code to make it presentable for new comers;
- providing test data for everyone to use to validate functionality;
- establishing governance for the software;
- engaging with users and contributors and listening to feedback and ideas;
- scheduling review points in the calendar.

OSS Watch Case Studies - http://www.oss-watch.ac.uk/resources/casestudies.xml

OSS Watch has many case studies covering a range of Open Source Software issues and projects. They offer great insight and are kept updated. One synthesis[79] of seven different OSS projects (including MailScanner, Apache Cocoon, Sakai and Moodle) identified three key learning points:

- continuity of effort and requirement/need is fundamental
- sustainable projects graduate to a service model from a project model, often involving commercial relationships

79 *Sustainability Study: A case study review of open source sustainability models*, M*etc*alfe, V1.0, April 2007.

- there are multiple maturity stages, decision points, and associated funding decisions and funding decisions should be tied to key success indicators such as adoption.

A.5.6 There are some points to factor in about the costs of this approach:

- Ensuring a sufficient maturity of software could add significant costs (*eg* taking a prototype to reusable software is often a factor of 10);
- Cultivation will involve a sustained effort to move to a more open development model;
- The costs and likelihood of eventual success are difficult to predict;
- If it is successful, it will spread costs over a larger number of individuals and organisations;
- The ideal outcome is that it becomes financially self-sustaining.

A.5.7 Some metrics or indicators to monitor to know how well this approach is proceeding could be designed around one or more of the following:

- OSS Watch Software Sustainability Maturity Model (*ie* the SSMM Level);
- the Community Roundtable's Community Maturity Model (*ie* which stage from Hierarchy to Emergent Community to Community to Network);
- size of user community: rocketing / increasing / a 'known' community / decreasing / plummeting (*eg* specifically the number of individual active users);
- spread of user community: internal / external / cross-domain;
- number and spread of contributors (*eg* number of individual contributors, or the number of contributions from specific communities);
- continued executability (*eg* percentage monthly uptime, or number of failures a month);
- continued compilability (*eg* binary yes/no for compilation failure, or number of compilation warnings).

Further information and useful resources

OSS Watch
http://www.oss-watch.ac.uk/resources/ssmm.xml
http://www.oss-watch.ac.uk/resources/howtobuildcommunity.xml
http://www.oss-watch.ac.uk/resources/researchinfrastructure-sustainability.xml

Producing Open Source Software: How to Run a Successful Free Software Project (by Karl Fogel)
http://producingoss.com/

Community Roundtable Community Maturity Model
http://community-roundtable.com/2009/06/the-community-maturity-model/

A.6 Hibernation (knowledge-centric)

A.6.1 Rather than sustaining your software as operational software, you may choose to hibernate it. You may choose hibernation when your software has come to the end of its useful life, but may need to resurrect it to double-check analysis or prove a result. Alternatively, there may not be a user community for your software, but you believe one will occur in the future. Hibernation allows you to preserve the knowledge about your software so that it can be resurrected in the future.

A.6.2 Hibernation can be a one-off process. Unlike sustainability, which requires a continuous investment of resources, the hibernation process can have a beginning and – importantly – an end. Preparing software for hibernation can be resource heavy, and if the software is never resurrected, you may feel that those resources were wasted.

A.6.3 Hibernation allows you to store software that you do not currently need, but it requires a significant – if short lived – investment of resources.

A.6.4 Specific activities within this approach are likely to include:

- reviewing and improving documentation;
- recording the significant properties of software;
- archiving the software along with all documentation;
- scheduling review points in the calendar.

A.6.5 If the software is already OSS then hibernation should be relatively straightforward, since there ought to be a code repository, up to date documentation and a means to contact user and contributors (if any).

Apache Attic - http://attic.apache.org/

"The Apache Attic was created in November 2008 to provide process and solutions to make it clear when an Apache project has reached its end of life. Specifically to be: 'responsible for the oversight of projects which otherwise would not have oversight; and be it further ... is not authorized to actively develop and release the projects under its oversight'

It is intended to:

- Be non-impacting to users

- Provide restricted oversight for these codebases

- Provide oversight for active user lists with no Project Management Committee

It is not intended to: Rebuild community; Make bugfixes; Make releases

[…]

Options [for leaving the Attic] are:

- Forking the project - we'll link to any forks which have been created so please let us know

- Restarting the community in the Apache Incubator

- Recreating a Project Management Committee for the project"

A.6.6 There are some points to factor in about the costs of this approach:

- At its simplest, hibernation involves documenting pseudo-code (*eg* publishing the algorithm in a research paper) – this is inexpensive;
- However, ensuring rigorous documentation, test data, *etc* is time consuming;
- There is a small ongoing cost to ensure discoverability, accessibility, *etc* of hibernated software and materials;
- The big advantage of hibernation is that it should significantly reduce future development costs.

Resurrecting old code

"Have you ever been haunted by an old open source package that you wrote once, published, and then forgot about?"[80]

One example where academic code has been properly archived is some audio analysis software developed at Australia's Commonwealth Scientific and Industrial Research Organisation (CSIRO). Maaate is a C++ toolkit to parse and analyse audio data in the compressed/frequency domain. The source code is available under the GNU General Public License. It was first developed and then hosted on a project page on the CSIRO Mathematical and Information Sciences division. Some years later the developer was contacted as the pages and code were being taken down. The developer had now left CSIRO but was "glad" to be contacted. She takes up the story: "Since it is an open source project, I have now resurrected the old pages at Sourceforge. They are available from http://maaate.sourceforge.net/. I have re-instated the relevant web pages and documentation and updated all the links. I discovered that we did some cool things then and that it may indeed be worth preservation for the future. I expect Sourceforge is up to the task."

A.6.7 Some metrics or indicators to monitor to know how well this approach is proceeding could be designed around one or more of the following:

- completeness of documentation (code, design, testing, *etc*) (*eg* percentage completion of the Significant Properties Framework);
- currency of programming language, middleware and operating environment (*eg* number of updates in the past year, number of updates planned for next year, time to end-of-support-life in months, total number of months past end-of-support-life, or total number of major new releases that supersede the version used);
- archive availability and resilience (*eg* that defined by Service Level Agreements, frequency of backup, or percentage up time in the last month);
- compilability and executability at review points (*eg* see those defined in Migration, above).

Further information and useful resources

Towards a methodology for software preservation (Brian Matthews)
http://escholarship.org/uc/item/8089m1v1.pdf

Appendix 6 of the Blue Ribbon Task Force final report
http://brtf.sdsc.edu

80 <http://blog.gingertech.net/2008/08/23/resurrecting-old-maaate-code/> accessed 6 October 2010.

A.7 Deprecation

A.7.1 If software lacks a community, the resources to continue or a developer, then the only alternative is deprecation. All effort invested into the software comes to an end, but, unlike hibernation, no effort is invested in preparing the software beforehand. In the future, if someone wants to use the software, they may not be able to find a stored copy and it might be expensive or impossible to resurrect the software.

A.7.2 Deprecation is easy to perform, but often marks the end of a software package's life and is typically only chosen when no other option is available.

A.7.3 Deprecation is effectively enforced technical preservation – but without the thought and preparation to have any confidence that it'll work as an effective preservation strategy.

A.7.4 If the software is then required, then you will need to provide, or buy in services for, software archaeology. This involves rescuing software from obsolete or damaged hardware, media and software environments – consider it an emergency recovery strategy. It may involve media recovery, for example if the media is heavily damaged, but more likely the real problem will be in understanding the code or binary. If you just have a binary then further information is probably necessary to determine what environment the software can be run on. If you have the code then at least you are able to adapt it to make it run, but the code may start off as being effectively unintelligible. It will be especially difficult to recover if the code is old, poorly documented, lacking the original build tools (compilers, makefiles, *etc*). If it's not your code, or you've switched to using another programming language, then clearly it will be harder still. And without having pre-planned test data it will be hard to get the assurance that the software runs as intended – critical if you're after perfect repeatability.

A.7.5 Specific activities within this approach are likely to include:

- deciding on a timeframe for deprecation;
- notifying users and contributors of the intent to deprecate;
- archiving the software along with all documentation.

A.7.6 Some metrics or indicators to monitor to know how well this approach is proceeding could be designed around one or more of the following:

- infrequency of user engagement;
- completeness of documentation (*eg* see those defined in Hibernation, above);
- archive availability and resilience (*eg* see those defined in Hibernation, above).

A.7.7 There are some points to factor in about the costs of this approach:

- There are short term costs in formally shutting down development;
- Depreciation generally assumes software has been superseded and no emergency recovery effort is needed.

Further information and useful resources

Apache Attic
http://attic.apache.org/