Digital Preservation and Curation
The Danger of Overlooking Software

7 February 2011
Neil Chue Hong, Director
N.ChueHong@software.ac.uk
What sort of software might need to be preserved?

- Digital curators need to make tough decisions over what digital objects should be preserved.
- Software is a digital object which may need preservation:
  - Software which is an intrinsic part of the data of digital object being preserved:
    - Enable continued access to data
  - Software which is a research output:
    - Achieve legal compliance
  - Software which has an intrinsic heritage value.
Resources for courses?

- Work captured by the JISC funded Software Preservation Study undertaken by Curtis + Cartwright and the Software Sustainability Institute has created a framework for defining software preservation.
- What are the useful pieces of information for the digital curation community?
Software is no longer easy to define, let alone preserve
What do we sustain:
- Workflow?
- Software that runs workflow?
- Software referenced by workflow?
What do we sustain:
- Map?
- Software that creates map?
- Software that uses map?

Novel reuse of public sector data
http://www.mysociety.org

The light, coloured areas of the map represent places where it’s faster to use public transport than to drive if you want to get to work in central Edinburgh by 9AM (centred on postcode: EH1 2QL)
Preservation Challenges

• Software often has complex dependencies
  ▪ Sensitive to changes in its environment
  ▪ May require expert knowledge outside of the team

• There may be many variations in many versions
  ▪ Which do you preserve?

• Software is increasingly transient
  ▪ Cloud computing and workflows
When, where, what, why and how?
Decisions, decisions

• There are several approaches we have identified that could be classed as software preservation

• The choice depends on a number of factors, which change through time
Approaches

- Preservation (techno-centric)
- Emulation (data-centric)
- Migration (functionality-centric)
- Cultivation (process-centric)
- Hibernation (knowledge-centric)
- Deprecation
- Procrastination (I'll-get-round to it eventually-centric)
Preservation

• Aim to preserve original hardware and software in same state
• Purpose:
  ▪ Achieve legal compliance
  ▪ Create heritage value
  ▪ Enable continued access to data
• Advantages:
  ▪ Clearly defined
  ▪ Can often change to Hardware Emulation at later date
• Disadvantages:
  ▪ Costly, especially when hardware fails
  ▪ Does not guarantee future access if dependent on other hardware/software (e.g. networking)
  ▪ Can be vulnerable to malicious attack
Emulation

- Emulate original hardware / operating environment, keeping software in same state
- **Purpose:**
  - Achieve legal compliance
  - Create heritage value
  - Enable continued access to data
  - Manage systems and services
- **Advantages:**
  - Easier to manage virtualised hardware
  - If emulation layer continues to be developed, software can continue to be run indefinitely
- **Disadvantages:**
  - Need all aspects of hardware to be emulated correctly, including flaws
  - If emulation layer ceases to be developed, back to technical preservation
Migration

• Aim to update software as required to maintain same functionality, porting/transferring before platform obsolescence

• Purpose:
  ▪ Achieve legal compliance
  ▪ Enable continued access to data
  ▪ Encourage software reuse
  ▪ Manage systems and services

• Advantages:
  ▪ Allows further development of software
  ▪ Enables access on other platforms

• Disadvantages:
  ▪ Requires continued effort for development
Cultivation

• Aim to keep software “alive” by moving to more open development model bringing on board additional contributors and spreading knowledge of process

• Purpose:
  ▪ Enable continued access to data
  ▪ Encourage software reuse
  ▪ Manage systems and services

• Advantages:
  ▪ Increases chances of further development of software
  ▪ Potential for better migration to other platforms

• Disadvantages:
  ▪ Requires more coordination
  ▪ Possibility for loss of control of direction
Hibernation

• Aim to preserve the knowledge of how to resuscitate/recreate the exact functionality of the software at a later date

• Purpose:
  ▪ Enable continued access to data
  ▪ Encourage software reuse

• Advantages:
  ▪ Useful when you have a known break in effort

• Disadvantages:
  ▪ Can be difficult to check if hibernation processes are rigorous until after it is too late
Procrastination

• Aim to do nothing
• Purpose:
  ▪ What purpose?
• Advantages:
  ▪ Comes naturally
  ▪ Very cheap
• Disadvantages:
  ▪ Not a valid preservation technique!
  ▪ May require software archaeology skills in the future
• A valid approach if software has been superceded
Key decision points

• You’re probably thinking about software preservation:
  ▪ When you are acquiring the related digital object
  ▪ When you have been approached for advice
  ▪ When you are curating a wider collection
  ▪ When you are drafting a preservation policy / research data management plan

• Do you need different tools at different stages?
  ▪ Or does it depend on the type of software?
How much access?

- Are you in contact with the owner of the code?
- Is the developer of the software still available?
- Do you have access to the source code?
- Do you have access to the hardware the software is running on?
- Are there licensing or rights management issues related to the software?
  - Will this change the cost model for preservation?
Retrieve, Reconstruct, Replay

• Preservation of software is only the “top” level
• How do you ensure the behavioral characteristics of the software are preserved?
• What are the “significant properties” of the software?
  ▪ Functionality
  ▪ Form
  ▪ Dependencies
  ▪ Ownership
  ▪ Interaction
  ▪ …
How exact do you need it?

- 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
- Is integrity important?
- What are you preserving and why?
  - Software, functionality, data, knowledge,...
Sustaining software through migration and cultivation
The Software Sustainability Institute

• A national facility for research software
  ▪ Providing services for research software users and developers
  ▪ Developing research community interactions and capacity
  ▪ Promoting research software best practice and capability

• Sustaining software by helping to negotiate the stages of the software maturity cycle
Digital preservation and curation
The danger of overlooking software

From preserving results from research, to storing photos for the benefit of future generations, the importance of preserving data is gaining widespread acceptance. But what about software? It's easy to focus on the preservation of data and other digital objects, like images and music samples, because they are typically the end products of the work that people wish to preserve. The software used to access the data is an overlooked middleman in the preservation process. Without the right software, it could be impossible to access the preserved data, which undermines the reason for storing the data in the first place.

This briefing paper is targeted at people who are responsible for preserving things like data and digital objects on behalf of others. These people typically work for libraries, museums and archives. Our goal in this paper is to explain why long-term software preservation is necessary, what needs to be understood before software can be preserved and how to get started.

When should software preservation be considered?

Software is used to create, interpret, present, manipulate and manage data. Software preservation should be considered when one or more of the following is true:

- The software can't be separated from the data or digital objects.
- In an ideal world, data can be isolated and preserved independently of the software used to create or access it. Sometimes this is not possible. For example, if the software and the data form an integrated model, the data by itself is meaningless so the software must be preserved. If the data isn't stored in an open, human-readable format, then the software is needed to access the data, so the software must be preserved.
- The software is classified as a research output.

In this case, the software falls under some Research Council's preservation policy. This means that the software must be preserved as a condition of its funding.

The software has intrinsic value

Software can be a valuable historical resource. If the software was the first example of its type, or it was a fundamental part of a historically significant event, then the software has inherent heritage value.

What are the issues?

Software presents some challenges to those who curate, preserve and archive. In particular, software preservation is difficult, because software is sensitive to changes in its environment. If there is a change to the computer or operating system on which the software runs, the software will often stop working properly. This change might not cause a complete software failure, it might only cause a subtle, yet important, change in results. Expert knowledge is needed to fully understand how a particular software component works and the effect that a change may have.

Software comes in many different forms, is written in a bewildering range of languages and can be licensed in a number of different ways. Additional difficulties can arise from the increasing use of web services and the cloud, where the software is a part of a larger ecosystem.

Congratulations! You've inherited some code

This briefing paper is targeted at software developers and project managers dealing with the issues of taking over code which they have not originally developed, where no proper handover occurred. It highlights some of the key pitfalls in working with inherited and "dusty deck" code and provides advice on the steps you should take. For advice on how to ensure that code is handed over properly, see the related briefing paper "Help! Your developer is running away!"

It is important to deal with this as early as you can so that you are prepared, especially if you are required to maintain a continued service.

Why is this important?

In many cases, issues arise when a piece of software is still being used long after the last developer who has any understanding of the code has left the community. This may be because of funding, changes in direction or simply because the software has been stable for a long time. The trigger is normally a change in the operating environment (e.g. the operating system changes) or a request for new functionality. Taking over code is often necessary and important because users continue to rely on the software for their work, whilst the software itself decays and becomes more prone to failure, lack of compatibility and then loses its point of no return. This can happen due to upgrades in the operating system, degradation of functionality within dependent libraries, changes in the underlying programming language development kit, etc. The kit goes on.

Organise it like an archaeologist

- Take an inventory
- Have you got all the code and documentation you can find?
- What development environment was used?
- Do you have any test cases?
- This will also give you a handle on the magnitude of the task.

Secure the site

- Shore up the version control system, put in place processes to build and test the code in a reliable and repeatable way.
- Assign specialists to each excavation task.
- Perhaps there are members of your team who have good experience with each piece of technology, if so, assigning them the task of working out the aspect could save a lot of time.

Prepare it for the next generation

- Once the code is cleaned up, make sure that the next generation of developers don’t have to do what you’ve just done.
- Update the documentation.
- Of course, if you have the resources, you might want to do a proper handover of the code.

Software Sustainability Institute
Software Preservation resources offered

• Software Preservation Study outputs:
  ▪ [http://www.software.ac.uk/resources/preserving-software-resources](http://www.software.ac.uk/resources/preserving-software-resources)
  ▪ Benefits Framework
  ▪ Briefing Papers

• Software Consultations
  ▪ Review of software / documentation

• Collaborative Projects

• Contact info@software.ac.uk