STFC

Tools and Guidelines for Preserving and Accessing Software as a Research Output

---

Report II: Case Studies

A Shaon, J Woodcock and E Conway

30 April 2009

Science & Technology Facilities Council e-Science

THE UNIVERSITY of York

Contents

1 Introduction

This report presents the findings from two case studies on the approaches of two contrasting organisations, the British Atmospheric Data Centre (BADC) and the University of York, to long-term maintenance and re-use of software artefacts. The primary objective of these case studies was to gain an understanding of the current best practice in software re-use over the long-term. Additionally, this exercise was envisaged as a means to gain insight into the outlook of the aforementioned organisations towards long-term preservation of software. The approach employed to conduct these case studies involved undertaking visits to these organisations in order to discuss with their software package and repository managers their approach to software maintenance and the ongoing problems of long-term preservation of software as well as how to accommodate change in the technological environment.

The remainder of this report is structured as follows:

Section 2 describes a case study on the BADC's approach to long-term preservation of software, analysing a number of its software tools and their preservation properties.

Section 3 presents a case study on the University of York that investigates the preservation and curation related aspects of two of their software products: Mondex and Z/Eves.

Section 4 presents and analyses the results of an exercise of applying the framework for significant properties of software (Report I) to the BADC Web Feature Service.

Section 5 discusses the issues identified from these case studies and explores possible solutions.

2 BADC Case Study

2.1 Background

The National Centre for Atmospheric Science's British Atmospheric Data Centre[1] (BADC) is a NERC Designated Data Centre which has the role: *to assist UK researchers to locate, access and interpret atmospheric data and to ensure the long-term integrity of atmospheric data produced by Natural Environment Research Council (NERC) projects.* This case study investigates the BADC's approach to long-term preservation of software by analysing a number of its software tools and their preservation properties.

The BADC currently has over 25TB of atmospheric data that are currently archived from 109 datasets for the consumption of over 6000 registered users[2]. The BADC also provides information and links to data held by other data centres. In many cases the BADC may be the only long-term archive of the data.

In order to facilitate efficient accessibility and usability of these large volumes of atmospheric datasets, the BADC also develops, supports and provides with access to a variety of software, which ranges from very simple data conversion tools to highly complex weather prediction software. Considering the importance of the BADC software in enabling accessibility and interpretation of its datasets, effective long-term preservation (i.e. re-use) of its datasets implies the need for appropriate preservation actions for its software[3].

2.2 Overview of BADC software

The different types of software that BADC currently provides and maintains are categorised in the table below. Each software package is described in the sections that follow.

Category	Description	Software Name
Data Discovery Software	Software which facilitates direct discovery and permit remote or local access to BADC data	Web Feature Service (Section 2.n)
Dynamic Data Processing Software	Software which processes archived data for the "on-the-fly" provision of processed data product	The BADC Trajectory Service, Data Extractor and GeoSplat.
Data Analysis Software	Generic Analysis tools	Xconvsh/convsh, GrADS and CDAT.

[1] http://badc.nerc.ac.uk/home/index.html
[2] http://www.noc.soton.ac.uk/coapec/pdfs/FM/P15Marsh.pdf
[3] This work undertaken in conjunction with the JISC funded SCARP project http://www.dcc.ac.uk/scarp/ which is conducting a wider study into the preservation needs of BADC.

Modelling Software	Large Scale Modelling software	Met Office Ported Unified Mode
Data Set Specific Software	Data Set Specific software tools and scripts which are informally archived	MST data plotting software
Other Software	Community based models and analysis tools.	

Table 2.1: Different Categories of BADC Software

The following sections examine key examples from the above categories exploring 1) the functionality the software provides users, 2) the human/technical dependencies and 3) their preservation status and requirements.

2.3 The BADC Trajectory Service

The BADC Trajectory Service provides a simple interface to an atmospheric trajectory model. In essence, the model is seeded with one or more *particles* at specific *locations* and *times*, and these particles are then traced forward in time according to four-dimensional wind fields resulting from a pre-computed atmospheric circulation model.

2.3.1 Functionality and Architecture

The BADC Trajectory service's browser-based interface operates in two stages. The first involves specifying the initial conditions for the trajectory run based on user inputs. The second is the calculation and dissemination of the trajectory results. Dissemination of the trajectory includes plotting the path of the specified wind parcel onto a global map (Figure 2.1) as well as plots of pressure, temperature and potential temperature.

The atmospheric trajectory model that underlies the trajectory service uses wind data that are from 40 years of archived data in a mixture grib and pp data formats held at the BADC[4]. The technical detail regarding the following types of processes and procedures is critical data provenance information for any trajectories generated:

- Observation processing;
- Data assimilation;
- Dynamics and numerical procedures;
- Physical processes;

[4] These wind datasets often called ECMWF winds as they are generated using detailed technical documentation about the Integrated Forecasting System (IFS) provided by the ECMWF (European centre for medium range weather forecasts).

- The Ensemble Prediction System;
- Technical and computational procedures.

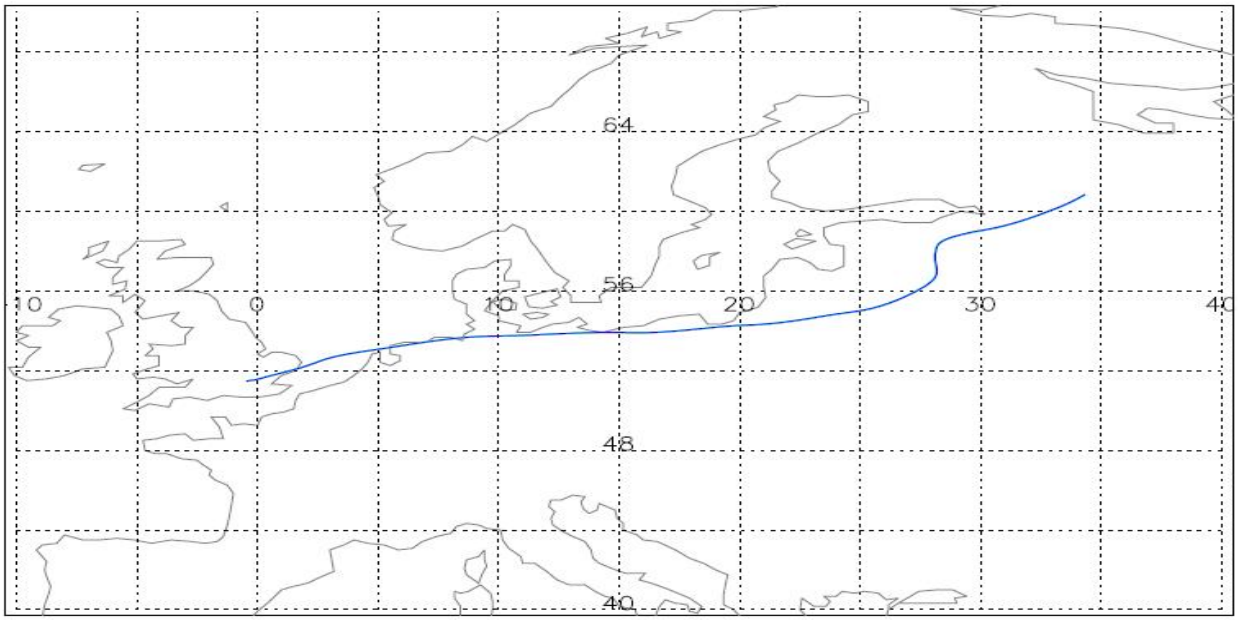

Figure 2.1: An Example Trajectory Path of a Wind Parcel

2.3.2 User and Technical Dependencies

The software is written in IDL with a perl web interface. It is currently well documented and supported by the BADC helpdesk but users will require some specialist knowledge in order to be able to use it.

2.3.3 Versions and Variants

The BADC has had only one version of the trajectories software which although the software author has left the BADC it is still capable of maintaining. However, the BADC has future interest in deploying the trajectory service on the UK National Grid to make it available to the UK Grid community. This would likely to require significant changes to be made to the software, thus resulting in a new version.

2.3.4 Preservation

The BADC Trajectory Service has the potential to be useful in both present and future atmospheric research. For example the output of the trajectory service could perhaps aid in emergency management by estimating the area affected by toxic gas dispersion. However, the BADC currently anticipates that this software has no real preservation merit in itself.

2.4 CDAT

CDAT (Climate Data Analysis Tools) is a suite of software subsystems and packages that collectively form an integrated environment for analysing climate science data. It was developed at the Program

for Climate Model Diagnosis and Intercomparison (PCMDI) using an open-source scripting language Python.

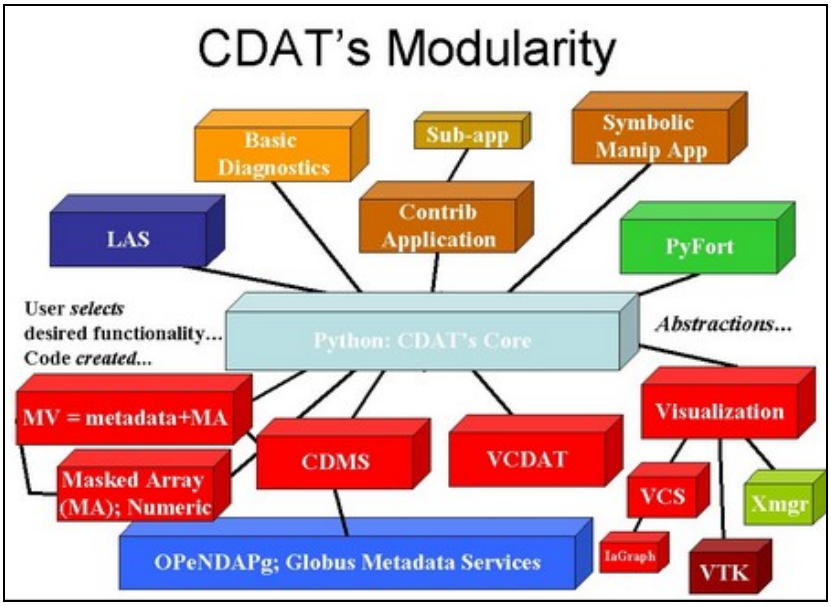

Figure 2.2: Dependencies of modules in CDAT

2.4.1 Functionality and Architecture

The underlying architecture of CDAT (Figure 2.2) consists of a number of software modules, such as a Climate Data Management System (file I/O, variables, types, metadata, grids) (CDMS), Climate Data Specific Utilities (spatial and temporal averages, custom seasons, climatologies) (CDUTIL) and Visualization and Control System (manages graphical window: picture template, graphical methods, data) (VCS). The software modules of CDAT provide the BADC users with an array of features and functionalities that include the followings:

- Plotting a variable from a file on a polar stereographic projection.
- Aggregating 1000s of files into one XML file so that slices of data can be read in across multiple files.
- Differencing 2 different datasets (as the Python Numeric package allows array algebra).
- Calculating departures from a climatology from a dataset.
- Regridding a dataset and then calculating a spatial average.
- Calculating the covariance between two variables.
- Calculating the mean and standard deviation of a variable.

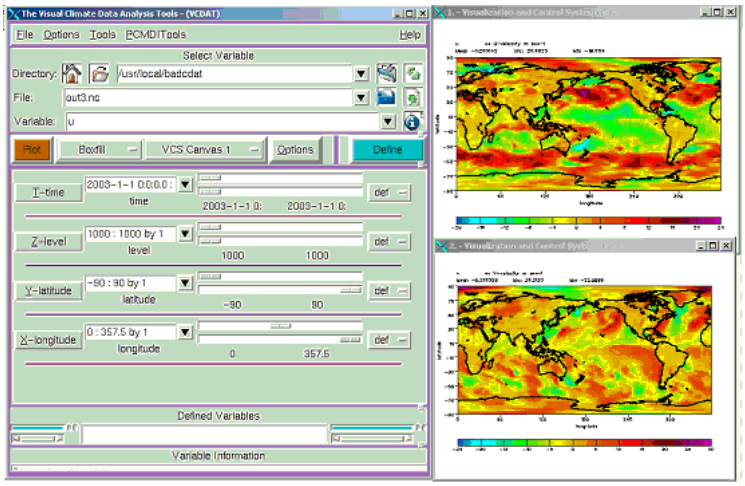

Figure 2.3: Screenshot of CDAT

2.4.2 User and Technical Dependencies

A user can potentially utilise CDAT in number of ways depending on their skill level and scientific objectives. For example, CDAT can be scripted to perform bespoke operation. Users can also use VCDAT Graphical User Interface (VCDAT) which is the graphical user interface for CDAT. It helps users become familiar with CDAT by translating every button press and keystroke into Python scripts. VCDAT does not require learning Python and the CDAT software. CDAT possesses a number of predefined analysis, conversion, sub-setting and array operations. It also has interfaces to FORTRAN and C/C++ allowing it to interact with user created models and programs.

One factor which has proven to be a barrier to its uptake has been the length of time in effort it takes to initially install CDAT. CDAT requires a Linux/Unix distribution. There are many different platform-specific operations that need to be carried out during the installation process, setting of environment variables, changing shell modes, installing libraries etc[5]. In order to remove these barriers a CDAT "Lite" is under development.

2.4.3 Versions and Variants

CDAT is fully supported on Macintosh OS X 10.4.x/10.3.x/PowerPC, RedHat Enterprise Linux WS 3.x/i386 and Enterprise Linux WS 4.x/i386. The BADC also provides support for porting the CDAT software to a number of other platforms, such as Sun/Solaris, SuSE Linux and other variants of Linux.

2.4.4 Preservation

According to the BADC, CDAT has considerable importance in facilitating analysis and interpretation of BADC's climate science data. Therefore, it is in BADC's interest to maintain support for CDAT over considerably long periods of time. However, BADC's current approach to maintaining and supporting CDAT may not suffice over the long-term as it may be necessary to make significant changes to the current versions of CDAT to ensure compatibility with future technological platforms. From an end user's viewpoint, it may be necessary to be able to verify consistency of its outputs across different versions and variants of CDAT. Therefore, it will be necessary to subject CDAT to long-term preservation in order to ensure its sustained usability.

[5] Full details of these complexities can be found at http://www2-pcmdi.llnl.gov/software-portal/cdat/download/installation-guide

2.5 Met Office Ported Unified Model

The Unified Model (UM) is essentially a suite of atmospheric and oceanic numerical modelling software developed and used at the Met Office. The model supports global and regional domains and a wide range of temporal and spatial scales that allow it to be used for numerical weather prediction as well as a variety of related research activities including climateprediction.net (a distributed project to consider a number of climate models to investigate the likely effects of climate change). The Ported Unifies Model software allows the Unified Model to be run on a user's own system.

2.5.1 Functionality and Architecture

Similar to CDAT, the Ported Unified Model software has a modular architecture that consists of the following components:

- **User Interface:** An X-Windows application for setting up model integrations. It comprises over 200 separate windows and may be customised by the user.
- **Reconfiguration**: A generalised interpolation package used to convert model data files to new resolutions and areas.
- **Atmosphere Model:** Grid point split-explicit dynamics and physical parameterizations.
- **Atmosphere-Ocean Coupling**: Software to run atmosphere and ocean models in coupled mode, including a dynamic sea-ice model.
- **Diagnostics and Output**: Internal model package to output a range of diagnosed and derived quantities over arbitrary time periods, sub-areas and levels.

2.5.2 User and Technical Dependencies

The UM is a large and complex software system, primarily designed for use in a research and operational forecasting environment. According to the met office, installation and/or use of the UM requires a good working knowledge of Unix and Fortran as well as experience in C Programming and Unix System administration amongst other things.

2.5.3 Versions and Variants

A ported version of the Unified Model, the Ported Unified Model (PUM), has been developed for running on workstations, PCs running the Linux Open Source operating system as well as the massively parallel computer systems used for Operational forecasting at the Met Office. The focus of most recent, work has been to optimise the Unified Model for use with vector supercomputers like the Met Office's NEC SX-8. This has been built upon previous work, including incorporating a non-hydrostatic dynamical core into the PUM. The latest release of the Ported Unified Model, 6.1, represents a significant upgrade of both the scientific and technical capabilities of the model. This included significant porting work to support the use of clusters of commodity-based computers.

2.5.4 Preservation and Management

A complex model such as the Unified Model is under continuous development by a large team of scientists and programmers. A source code configuration management system is used ensure successful coordination of these developments. Source code developments, or modification sets as they are known are under the overall control of the Unified Model system manager and day-to-day control of a code librarian. In addition, a proprietary source code revision system is used to merge new code with the development stream, but plans are underway to automate the process further using modern source code control software tools that will be portable across computer platforms.

The numerical weather prediction capability of the UM has significant potential usefulness both at present and in the future. However, the highly complex underlying architecture coupled with volatility of the state of UM is likely to make it vulnerable to rapid changes in computer related technology. In the unfortunate event, should the development and maintenance of the UM cease in the future, it is likely to be come unusable over time, unless the software itself along with sufficient information about its different properties, such as its use, installation and performance are preserved over the long-term. A potential challenge of long-term preservation of the UM could be ensuring adequacy, consistency and accuracy of documentation across all its versions and variants, especially due to the large number of developers and users involved. However, the BADC (or the Met Office), under its current remit, has no long-term preservation plan or strategy in place to ensure longevity of the UM.

2.6 BADC Web Feature Service (WFS)/GeoServer

The British Atmospheric Data Centre (BADC) provide a Web Feature Service (WFS), which, in general, enables retrieving and updating geospatial data encoded in Geographic Markup Language (GML)[6], or any GML-based formats, irrespective of the location or storage media of the data.

2.6.1 Functionality and Architecture

The architecture of this system is primarily composed of two instances of GeoServer[7] deployed as the Open Geospatial Community (OGC) WFS 1.0[8] compliant web services over two contrasting datasets residing at the Centre for Ecology & Hydrology (CEH), Lancaster and the British Atmospheric Data Centre (BADC) respectively.

GeoServer used for the BADC WFS, is an open source Java-based web server that provides a suitable means of promoting and publishing Geospatial information on the web using various OGC standards. Being an open source and community-driven product, GeoServer has an increasingly large and diverse user and developer community from around the world. This has, in effect facilitated easily available user support and accelerated error-fixes and feature improvements. In addition, by providing OGC standards compliant software, GeoServer has established itself as one of the most commonly used software products within the Geo-spatial community.

Since its release in 2003, GeoServer has undergone a number of version changes. Although a standard version of GeoServer provides a suitable facility for serving up geospatial data in pure GML, it is not possible to do the same for formats that are based on GML - a principle requirement for the BADC WFS. Therefore, implementation of the BADC WFS used the "Complex DataStore" version of GeoServer, which enables representation of data from a relational database in a GML-based application schema (e.g. Climate Science Modelling Language[9]) that is defined independently of the underlying database structure. This special edition of GeoServer was a research endeavour by SeeGrid[10] with contribution from the GeoServer community and the BADC. The functionality of this GeoServer has wide applicability for harmonising disparate data sources through a common data model – a common challenge in many data management domains that has no uniformly effective solution to date.

---

[6] Geography Markup Language is an XML grammar written in XML Schema for the description of application schemas as well as the transport and storage of geographic information - http://www.opengeospatial.org/standards/gml

[7] http://geoserver.org/display/GEOS/Welcome

[8] OGC Web Feature Service - http://www.opengeospatial.org/standards/wfs

[9] http://ndg.nerc.ac.uk/csml/

[10] https://www.seegrid.csiro.au

2.6.2 User and Technical Dependencies

Installation of the Complex DataStore Geoserver from WAR (Web Application Archive) file is relatively uncomplicated and does not require the users to have any understanding of the underlying programming language. The GeoServer WAR file can be deployed under any Java-based Web Server, such as Tomcat. Configuring the software for particular database and output format, on the other hand, is complex and requires users to have considerable knowledge about XML, XML Schema, SQL and GML. The BADC wiki[11] has detailed instructions about how to configure GeoServer.

2.6.3 Versions and Preservation

The Complex DataStore version of GeoServer is not a part of any of the standard releases of GeoServer and no longer actively supported by the GeoServer community. It is currently available as a set of unsupported source code modules in the GeoServer code base and in the form of WAR files at the SeeGrid website. In either case, continuous maintenance and support for this software is not guaranteed, and without long-term maintenance and preservation, it is likely to become unusable over time. Even if the source code is still around, lack of sufficient information about re-building the software from source code, and how to configure and use it once re-built, could make its revival and re-use less effective. From this perspective, the Complex DataStore version of GeoServer provides a good example of a software package that needs to be preserved and curated over the long-term.

Further, in its current state, the BADC GeoServer is used for serving up only a small subset of the UK Met Office MIDAS[12] dataset, which is the largest dataset handled by the BADC. However, BADC has plans for (re-)deploying their GeoServer on a larger scale with the full MIDAS dataset in the near future. Considering this, the BADC should benefit from efficient long-term preservation of this software, though it is not within their current remit.

[11] http://proj.badc.rl.ac.uk/ndg/wiki/GeoManual
[12] http://badc.nerc.ac.uk/data/ukmo-midas/

3 YORK Case Study

3.1 Background

The York case study has two aspects. The first involves the preservation of the specification and design artefacts for a secure smartcard application: Mondex, originally developed in the early 1990s by National Westminster Bank. The application is designed to work like electronic cash, suitable for low-value cash-like transactions, with no third-party involvement and no cost per transaction. Once the issuing bank has loaded electronic cash into the system, it has very little further control over it: just like real cash. Because of the lack of third-party control, it is crucial that the security of the card cannot be broken; otherwise criminals could electronically "print" money with ease.

The target platform smartcard (state-of-the-art in the early 1990s) had an 8-bit microprocessor, a low clock speed, limited memory (256 bytes of dynamic RAM, and a few kilobytes of slower EEPROM), and no built-in operating system support for tasks such as memory management. Also, power could be withdrawn at any point during the processing of a transaction, if say the card was withdrawn from the reader. The engineers had to design and implement the secure cash-transfer protocol under these severe space and speed constraints.

The bank developed the full Mondex product to one of the very highest standards available at the time: ITSEC Level E6. This mandates stringent requirements on software design, development, testing, and documentation procedures, and also mandates the use of formal methods to specify the high-level abstract security policy model, to specify the lower-level concrete architectural design, and to provide a formal proof of correspondence between the two levels, in order to show that the concrete design enjoys the abstract security properties.

Formal methods are mathematically based techniques for the specification, development, and verification of software and hardware systems. The use of formal methods for software and hardware design is motivated by the expectation that, as in other engineering disciplines, performing appropriate mathematical analysis can contribute to the reliability and robustness of a design. However, the high cost of using formal methods means that they are usually only used in the development of high-integrity systems, where safety or security is important. The formal method used in the original development was the Z notation, although the case study involved additional methods: Alloy, ASM, Event-B, OCL, PerfectDeveloper, pi-calculus, and Raise.

The second aspect of York's case study is the curation of the main tool used: Z/Eves.

3.2 Overview of York's Software and Tools

York works mainly with the Z/Eves system, originally developed by ORA Canada. This is itself a legacy system, no longer under development and maintenance, although York is collaborating with one of the original designers on modernising and extending the system. Other tools include the CZT (Community Z Tools), the Rodin Platform, and PVS. These tools allow us to undertake deep semantic analysis of mathematical documents.

3.3 Z/Eves

3.3.1 Functionality and Architecture

Z/Eves is a simple and user-friendly tool for proving theorem. In effect, it is an implementation of the international standard for Z. It provides facilities for constructing specifications, with a GUI-based editor with pictorial support for the mathematical symbols characteristic of Z. It interfaces to standard document-preparation tools: Word, LaTeX, and Postscript. It allows users to check the consistency of their specifications at four levels:

1. ***Syntactic correctness:*** The basic grammar of expressions and paragraphs can be automatically checked. Good feedback is provided on incorrect syntax.

2. ***Type correctness:*** Documents can be automatically checked to ensure that they are mathematically well-typed. That is, that every expression has the right kind of value for its context. This ensures that, for instance, floating-point numbers cannot be used implicitly where integers were expected. Good feedback is provided on ill-typed documents.

3. ***Definedness:*** Documents can be checked to make sure that every expression is well defined. Examples of undefined expressions include: the result of division by zero, the first element of an empty list, and the largest prime number. As these examples suggest, undefinedness can arise from simple or quite sophisticated concerns, and in general a mathematical theorem must be proved to demonstrate definedness. Although the statement of the theorem required is generated automatically, its proof may require guidance from the user. It is possible to make the proof trivial by adding the statement of the theorem to the specification. The problem then becomes one of consistency.

4. ***Consistency:*** Documents can be checked to make sure that they are mathematically consistent. This amounts to finding a mathematical object that has all the properties required by the specification. Once more, the statement of the mathematical theorem can be derived automatically, but in general its proof cannot, and it will require guidance from the user.

The toolsets York uses each supports a similar kind of mathematical document, but there are some small but significant differences. The syntax and type rules vary between different dialects of Z, and between these and the main alternative notations of B and VDM. More significantly, the nature of definedness varies dramatically (with VDM even using a three-valued logic), and although the notion of consistency is broadly the same, there are some important differences in specific details.

3.3.2 User and Technical Dependencies

The installation of Z/Eves is straightforward, following the guidance given in the *Software Manual for Unix Z/EVES Version 2.1*, TR-97-6028-01e, by Irwin Meisels (Release date June 1996, Latest revision date July 2000). Installation additionally requires Python and Tcl/Tk. Currently, however, the tool is no longer distributed due to licensing problems (although versions are still advertised for downloading on several sites). York is working with one of the original designers of Z/Eves to produce an updated open-source version of the system. This new version retains the open-source front-end to the system, with its Z-specific tooling, but replaces the proprietary underlying reasoning engine. The new back-end takes advantage of modern, more sophisticated algorithms for automated reasoning.

Problems have been reported in installing the required mathematical fonts under certain versions of Linux, but good advice on this specific matter is available on the Web.

High-quality documentation requires a LaTeX system, rather than Word. The tetex system is usually distributed with most versions of Linux, but Windows users will probably have to install LaTeX themselves. Tex Live has a comprehensive version of LaTeX and is available for free download from www.tug.org.

3.3.3 Versions and Variants

The most recent version of Z/Eves was 2.3.1, which was released in June 2004.

3.3.4 Preservation and Management

3.3.4.1 Preservation of Z/Eves Software

Z/Eves is no longer distributed due to licensing problems. Indeed, the company responsible for the original development (ORA Canada) no longer exists. Binary versions of several releases of Z/Eves exist, including the most recent version, 2.3.1, and existing licences permit continued use. It is still possible to download binary copies from websites set up before distribution officially stopped, although the legality of downloading from these sites must be highly questionable.

The Mondex specification takes Z/Eves (GUI 1.5, Python 2.4) 2sec to parse and typecheck and the proofs scripts take 11min 20sec to run to completion. This is on a Tablet PC with a dual Pentium T2400 CPU, running at 1.83GHz with 2GB RAM 2005 under Windows XP SP2 with 48% CPU load. An experiment was undertaken to compare the performance of the Z/Eves tool on modern hardware with a state-of-the-art PC available when the Mondex application was being developed. This was successful because it was possible to find a PC from 1996, complete with its Windows operating system and a binary copy of Z/Eves. The proof scripts take more than six hours to run to completion.

3.3.4.2 Preservation of Application Software

The Z specification and refinement documents for the Mondex application existed on paper (published in 2000 as an Oxford University monograph). Electronic copies of the documents were obtained from the authors and made available through SourceForge. These documents were marked up using LaTeX, and had originally been parsed and type-checked using the Fuzz checker, but deeper semantic analysis for definedness and consistency had been conducted only by hand. An important aspect of the preservation of these documents was to carry out an automation of these analyses, and to make the corresponding scripts available for others. During this work, the York preservation team discovered and corrected some errors in the sources.

York considered how to curate the Mondex specification and design documents, and identified an important requirement: to re-express or translate the intellectual content as models in alternative specification notations. They successfully remodelled Mondex in seven different notations: Alloy, ASM, Event-B, OCL, PerfectDeveloper, pi-calculus, and Raise, and this has yielded some interesting comparative benchmarks. Recently, they have started to make some systematic translations between these different notations. They have some theoretical results to underpin inter-working between different logics, and expect to be able to develop some sophisticated trusted translators in future projects.

4 Applying the SP framework to the BADC Web Feature Service/GeoServer

This section details the outcomes of an exercise of mapping the Framework for Significant Properties of Software to the BADC WFS/GeoServer. It first presents the mapping itself and then discusses the lessons learned from the activity.

4.1 Package Properties

Property Category	**Software Property**	
	Name	**Value**
Functionality	**Purpose**	Enabling publishing and querying of Geospatial data on the web using open standards
	Keyword	Web feature service
Provenance and Ownership	**package_name**	GeoServer
	Owner	GeoServer and SeeGrid
	Licence	GNU GENERAL PUBLIC LICENSE Version 2 http://geoserver.org/display/GEOS/License
	Location	http://geoserver.org/display/GEOS/Welcome
Software Architecture	**Overview**	The software architecture is comprised of a series of modules for handling requests for geospatial data as geographical features across the web using platform-independent calls, such as HTTP Get and Post and SOAP. http://geoserver.org/display/GEOSDOC/1+GeoServer+Architecture
Software Composition	**software overview**	http://geoserver.org/display/GEOS/What+is+Geoserver
	Tutorials	Installaton: http://geoserver.org/display/GEOSDOC/1+Getting+Started
	requirements	Operating system: Window/Linux/Unix Minimum RAM: 512 megabyte Java 1.5 or higher

4.2 Version Properties

For the community schema version of GeoServer, we can get the following properties.

Property Category	**Software Property**	
	Name	**Value**
Functionality	**functional_description**	1. Enables definition and description of supported datasets as geographic features. 2. Queries datasets as geographic feature based on user specified queries in the OGC standardised query format and returns the result set in GML or GML-based format according to the definition of the feature requested.
	release_notes	Not a standard release
	Algorithm	http://proj.badc.rl.ac.uk/ndg/wiki/NERCGeoServer
	input_parameter	1.http://www.opengeospatial.org/standards/wfs 2.http://proj.badc.rl.ac.uk/ndg/wiki/GeoManual
	output_parameter	GML or any GML-based application schema
	Interface	http://proj.badc.rl.ac.uk/ndg/wiki/GeoManual
	error_handling	FAQ: Not available Bug reports at : not supported
Provenance and Ownership	**version_identifier**	1.6 (Community Schema/Complex DataStore)
	Licence	As package
Software Environment	**programming_language**	Java
	hardware_device	Platform Independent
Software Architecture	**detailed_architecture**	http://geoserver.org/display/GEOSDOC/1+GeoServer+Architecture
	dependent_package	Tomcat 5.5 or higher - http://tomcat.apache.org/download-55.cgi
Software Composition	**Source**	Source code modules for this version at http://svn.geotools.org/geotools/branches/2.4.x https://svn.codehaus.org/geoserver/branches/1.6.x
	Manual	Manual at: User Manual: http://proj.badc.rl.ac.uk/ndg/wiki/GeoManual

	Installation	Installation, at : https://www.seegrid.csiro.au/twiki/bin/view/Infosrvices/GeoserverCommunitySchemasDownloads?skin=clean.nat%2cpattern Build at: https://www.seegrid.csiro.au/twiki/bin/view/Infosrvices/Geoserver16DevelopmentSetup
	test_cases	samples at https://www.seegrid.csiro.au/twiki/bin/view/Infosrvices/GeoserverGeoscimlTestbed3Downloads
	Specification	http://geoserver.org/display/GEOS/Complex+Datastore https://www.seegrid.csiro.au/twiki/bin/view/Infosrvices/GeoserverCommunitySchemasDownloads?skin=clean.nat%2cpattern

4.3 Variant properties

The Community Schema version of GeoServer is platform independent and does not have any variants. However, for an instance of this software on a Linux x86-64 operating system, we can outline the following properties.

Property Category	**Software Property**	
	Name	**Value**
Functionality	**variant_notes**	Platform independent
Provenance and Ownership	**Licence**	as package
Software Environment	**Platform**	x86 architecture
	operating_system	Linux x86-64
	Compiler	Java 1.5 or higher
Software Architecture	**dependent_package**	Tomcat 5.5 or higher
	memory_usage	-512 MB RAM
Software Composition	**Binary**	Geoserver-community-schemas.war
	Source	http://svn.geotools.org/geotools/branches/2.4.x https://svn.codehaus.org/geoserver/branches/1.6.x
	configuration	https://www.seegrid.csiro.au/twiki/bin/view/Infosrvices/GeoserverCommunitySchemasDownloads?skin=clean.nat%2cpattern

4.4 Instance properties

We then assume for a specific installation we would give the following information.

Property Category	Software Property	
	Name	**Value**
Provenance and Ownership	**Licensee**	*user x*
	Conditions	see licence
	licence_code	None
Software Environment	**environment_variable**	JAVA_HOME, JRE_HOME and GEOSERVER_DATA_DIR set to a specific path JAVA_OPTS set to *-Xms512m -Xmx512m*
Software Composition	**File**	Location of WAR file on machine.

4.5 Lessons Learned

The primary objective of this exercise of mapping the framework for significant properties of software to GeoServer was to evaluate the overall efficiency of the framework in terms of the following criteria:

1. Degree of relevance of the software properties captured by the framework to the software that it is applied to.
2. Adequacy of the information recorded by the framework, i.e. whether the information captured is sufficient or insufficient or excessive.
3. Ease of use, i.e. how easy or difficult it is to apply the framework to the software. This takes into account the level of knowledge required about both the framework and the software used for the mapping task.

The experience from this exercise highlights that the framework is indeed relevant to the software (i.e. GeoServer) used as well as being adequate in terms of the information recorded. However, this exercise also indicates that it is necessary to have considerable understanding of both the framework and software in question to accurately map the framework to the software.

In particular, it would be difficult to distinguish between the four different categories of significant properties of software (e.g. Package, Variant etc.) captured by the framework, without understanding their underlying notions in the context of the framework. For example, a typical user would consider a software under his/her possession as a software package, while the framework actually refers to it as a "Instance" (a copy of the software on user's machine) and the software, as generally advertised by its vendor, as a "Package".

Furthermore, values for some of the properties recorded by the framework, such as "software architecture" "dependent library" and "source code" would unlikely to be known or even meaningful

to a novice user of the software; significant knowledge about the underlying software architecture is required for such a task. For example, long-term preservation of an AutoCAD design document may also require the preservation the AutoCAD software used to create the document. In such a case, even an expert user of such software would be unlikely to have any considerable knowledge of the underlying architecture and/or implementation related aspects of the software, hence would be unable to apply the framework effectively.

A possible solution to the aforementioned issue with the framework would be to provide the users of the framework with suitable tool(s) for creating/recording significant properties of software. Ideally, such tools would facilitate the creation of significant properties by providing the users with appropriate guidelines (e.g. explanation of significant properties, tutorial, etc.) in a user-friendly manner. In addition, automating the tool as much as possible (while ensuring the accuracy of the information recorded) should also help improve the user experience and overall efficiency of the framework.

5 Discussion

5.1 BADC's Approach to Software Preservation

It is evident from the analysis of different BADC software detailed in this report that the BADC, at present, has no considerable long-term preservation plans or strategy for its software and tools. It considers the long term archiving of software an impractical option principally due to the complex dependencies of software. It takes the view that it expects much current software will be superseded by newer software which will be capable of recreating and enhancing much of the existing analysis and access functionality. There will however be data set specific analysis models based in the user community for which the BADC anticipates that this will not happen. Further, the cost of archiving such models by migrating to new technologies as they evolve is prohibitive and emulation technologies have not yet matured sufficiently to allow confidence that storage of binary executables will be sufficient for preservation purposes. The BADC additionally considers it to be outside their core remit to harvest and archive such models.

5.2 York's Approach to Software Preservation

The York approach to preservation is illustrated by the Mondex and Z/Eves case studies. For Mondex, the issue was to preserve the intellectual content of the documentation. This was tackled in two ways. The first was to make as few changes as possible to the original source material, which was written in a mixture of Z and English. Some minor changes were made to correct some errors in the original specifications and design that were detected during the case study. This included not just syntax and type errors, but some deeper semantic errors too. As well as now being publicly available, this means that the documentation is more accurate than it was before the investigation. A measure of success is that there are a number of users continuing to work on the documentation in further experiments.

The second way of preserving the intellectual content of the documentation was to translate it into variety of different mathematical languages. This entailed some major changes, as the work was not carried out automatically. The resulting models have some stark differences. But, interestingly, they also have some striking similarities, and there is good reason to believe that the overall functionality is

preserved. These new languages have allowed Mondex to be analysed with a much wider collection of tools.

The work should be made more systematic, and this will be the subject of future research projects. York has plans to build tools to translate between the mathematical languages used in this kind of work. These translators must rely on theoretical results for their soundness. The theory required must link different sorts of mathematical logic and different presentations of the discrete mathematical theories of sets, relations, functions and so on. It needs to address different modelling and structuring techniques, as well as the program development methods used in each language. Noticing the difference in modelling adopted in each language, methods for refactoring models will be needed after translation. These are tough mathematical challenges.

The second part of the case study involved the preservation of a particular tool: Z/Eves. The major issue to be tackled here is not a technical one, although its solution is. The problem was a change in the licence arrangements for using Z/Eves. York's solution is to re-implement the proprietary component and to make the replacement freely available. This solution is not ideal, since it requires a considerable amount of effort. It also introduces problems of backwards compatibility. The component is Z/Eves' reasoning engine, and since the theorems it is trying to prove are in general undecidable, it uses a sophisticated set of heuristics. It is difficult to reproduce these heuristics, and indeed better ones are known today. So, although York does not expect the replacement to be any more or any less sound, the level of automation will change. So some Z/Eves proof scripts that worked before may not work with the new system.

York envisages all this as a fertile area for future research.

5.3 Conclusions

Recent times have witnessed a significant increase in the awareness of the necessity of long-term preservation of data, and the work done in this area. Long-term preservation of software, on the contrary, is a relatively underexplored topic of research and there is little practical experience in the field of software preservation as such. This is mainly due to the inherent complexity of software (in terms of dependencies between its components), which poses a significant barrier to its long-term preservation. The case studies presented in this report provide evidence of this. The BADC, for example, regards the long term archiving of software as an impractical option due to the complex dependencies of software. The case studies also illustrate the complexity of the task of long-term software preservation. For example, the York's approach to preserving Mondex entails highly complex translation of the intellectual content of Mondex documents into different mathematical models, and manual and onerous error checking of the original source materials (i.e. the Z specification).

In addition, the BADC case study highlights a prohibitive factor for long-term preservation of software: *cost of preservation*. Effective preservation of a digital object over the long-term requires its continuous management and enhancement over its lifecycle. This involves, amongst other tasks, periodically assessing (and improving) the adequacy of the preservation strategy for ensuring effective re-construction and re-use of the digital object notwithstanding any related technological changes. This is expected to impose significant recurring costs on the organisation undertaking long-term preservation, in terms of technical resources, efforts etc. required. There likely to be even greater efforts and hence costs required for the long-term preservation of software due to its inherently complex nature, i.e. the complex dependencies between its components.

Thus, for an organisation, such as the BADC who is already bearing the costs of maintaining and developing a wide array of software, it is difficult to justify and incorporate within its current remit and budget, the additional costs of long-term software preservation as such an activity might not be deemed beneficial to BADC in the short-term. For example, the BADC trajectory service is efficiently maintained and sufficiently documented for its current use; dedicating additional efforts (e.g. recording and preserving metadata, devising preservation mechanism etc.) for its long-term preservation is not likely to hold any tangible short-term benefit for BADC. In addition, the high complexity and costs of employing currently available preservation mechanisms, such as migration and emulation would also add to the overall costs of long-term preservation of software. Furthermore, there are currently no suitable cost models for long-term digital preservation, mainly owing to the uncertainties associated with such a task, which impart a high degree of variance to any attempt to estimate its cost. This in effect makes the task of preserving software over the long-term even less lucrative to an organisation that may already be burdened with the responsibility of pursuing other short-term goals.

Considering the findings of the case studies presented here, both the BADC and York should benefit from suitable software preservation models and tools for effectively addressing the complexity of software and the issues (e.g. costs) associated with its long-term preservation. For organisations, such as the BADC who is actively undertaking software developments and maintenance, it should be possible to integrate such preservation models and tools with existing systems for software developments and maintenance. For example, models may be developed for extending existing short-term software repositories to incorporate long-term preservation, and tooling can be provided to annotate software with its significant properties during its development from within its development environment. The framework for software preservation presented in ***Report I*** and the software significant properties editing tool, SPEQS in ***Report III*** should be considered as steps in that direction. However, further work is required in this area.