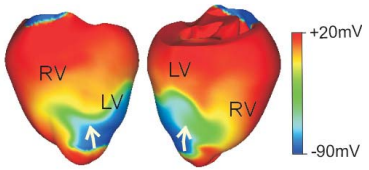
A Framework for the Significant Properties of Software

Brian Matthews, Brian McIlwrath,
Esther Conway, David Giaretta

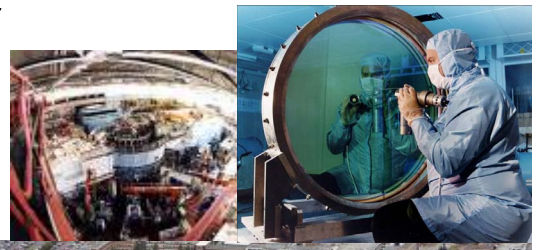
07/04/2008

Science & Technology Facilities Council
e-Science

Science and Technology Facilities Council



- Provide large-scale scientific facilities for UK Science
  - particularly in physics and astronomy
- E-Science Centre – at RAL and DL
  - Provides advanced IT development and services to the STFC Science Programme
  - Strong interest in Digital Curation of our science data
  - Keep the results alive and available
  - R&D Programme: DCC, CASPAR



Study into the Significant Properties of Software for Preservation.

- Software very large topic
  - Diversity in application of software
  - Diversity in software architecture
  - Diversity in scale of software
  - Diversity in provenance
  - Diversity in user interaction
- Need to limit scope
  - Scientific and mathematical software
  - Limited commercial consideration
  - Limit consideration of user interaction
- Finding information
  - Literature
  - Talking to developers of packages and software repositories
    - Starlink, BADC, CCPForge, NAG, etc.
    - Experience in maintaining and distributing software over a long period.
    - Accommodating change in software environment
- Developing a framework for software properties.

Software Preservation

- What is software preservation?
  - Storing a copy of a software package”
  - Enabling its retrieval in the future
  - Enabling its reconstruction in the future
  - Enabling its execution in the future

Not what most software developers and maintainers do.

Why Preserve Software ?

- Museums and archives:
  - Either supporting Hardware
    - E.g. Bletchley Park, Science Museum,
  - Or in its own right
    - Chilton Computing, Multics History Project
- Preserving the work
  - E.g. research work in Computing Science
  - Reproducible
- Preserving the Data
  - Preserving the software is necessary to preserve other data
  - Keep the data live and reusable
  - Prime motivation for STFC
- Handling Legacy
  - Specialised code from the past which still needs to be used
  - Usually seen as a problem!

Significant Properties of Software

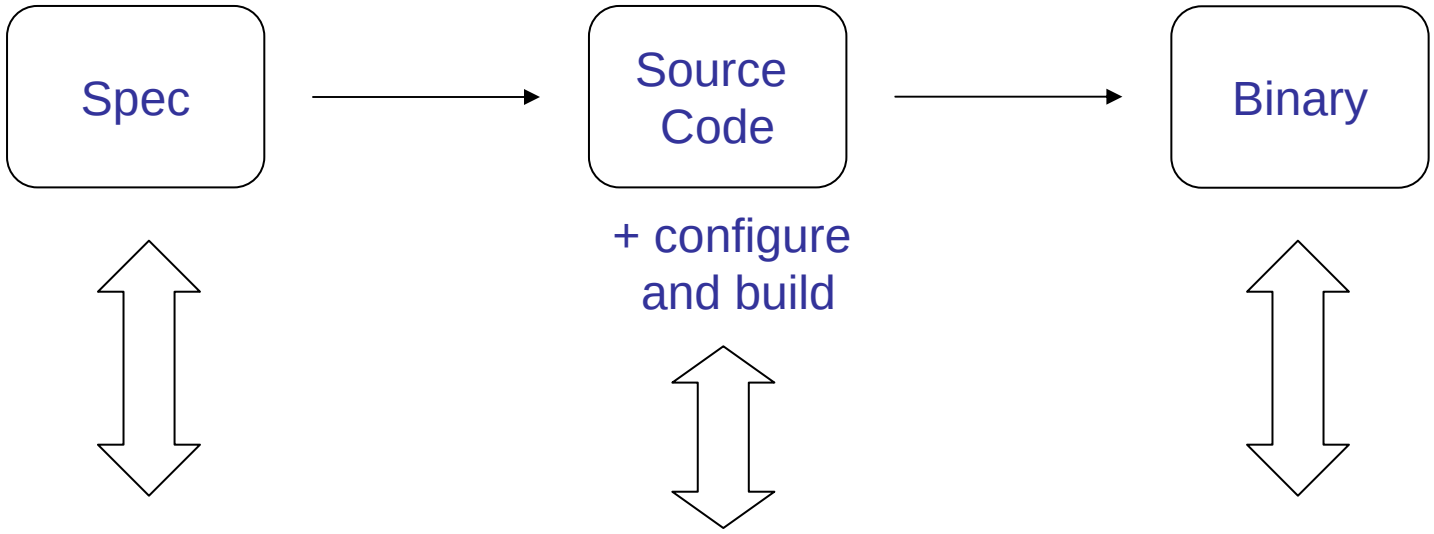
***Significant properties**, are essential attributes of a digital object which affect its appearance, behaviour, quality and usability.*

- What to attributes do we need to take into account?
  - Functionality
    - what it does and what data it depends on
  - Environment
    - platform, operating system, programming language
    - versions
  - Dependencies
    - Compilation dependency graph
    - Standard libraries
    - Other software packages
    - Specialised hardware
  - Software is a Composite digital object
    - Collection of modules
    - Specifications, Configuration scripts, test suites, documentation
  - Architecture
    - Client/server, storage system, input / output
  - User interaction
    - Command line, User Interface
    - User model

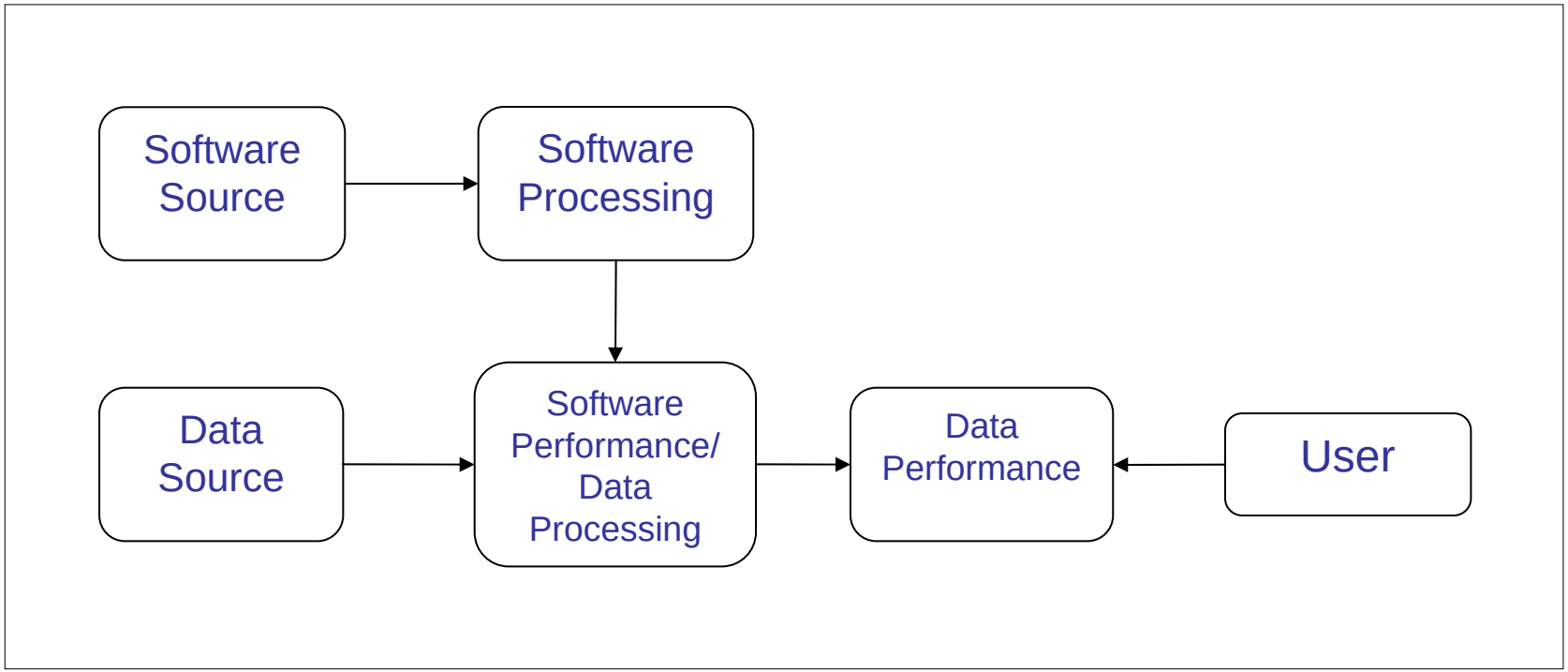
Clearly Software is highly complex with a lot of factors which need to be considered

we need a framework to organise and express software.

Preservation Approach and Software Process

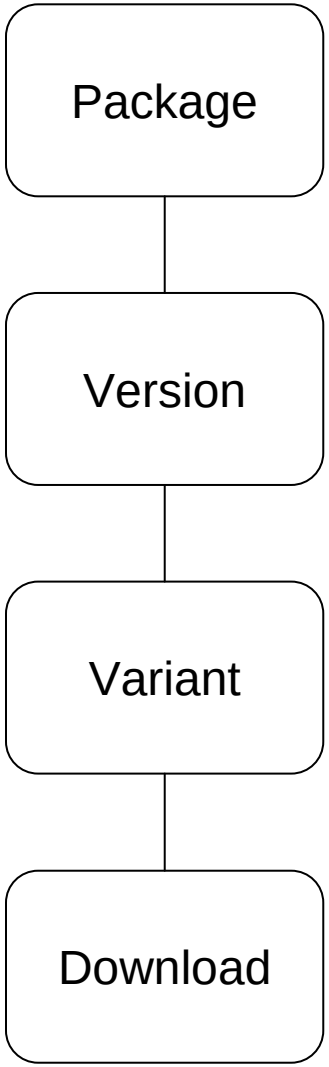


Performance Model for Software



- Testing data performance to judge adequacy of the software performance.
- Important to maintain software test suite to assess preservation of significant property.

A Framework for Software

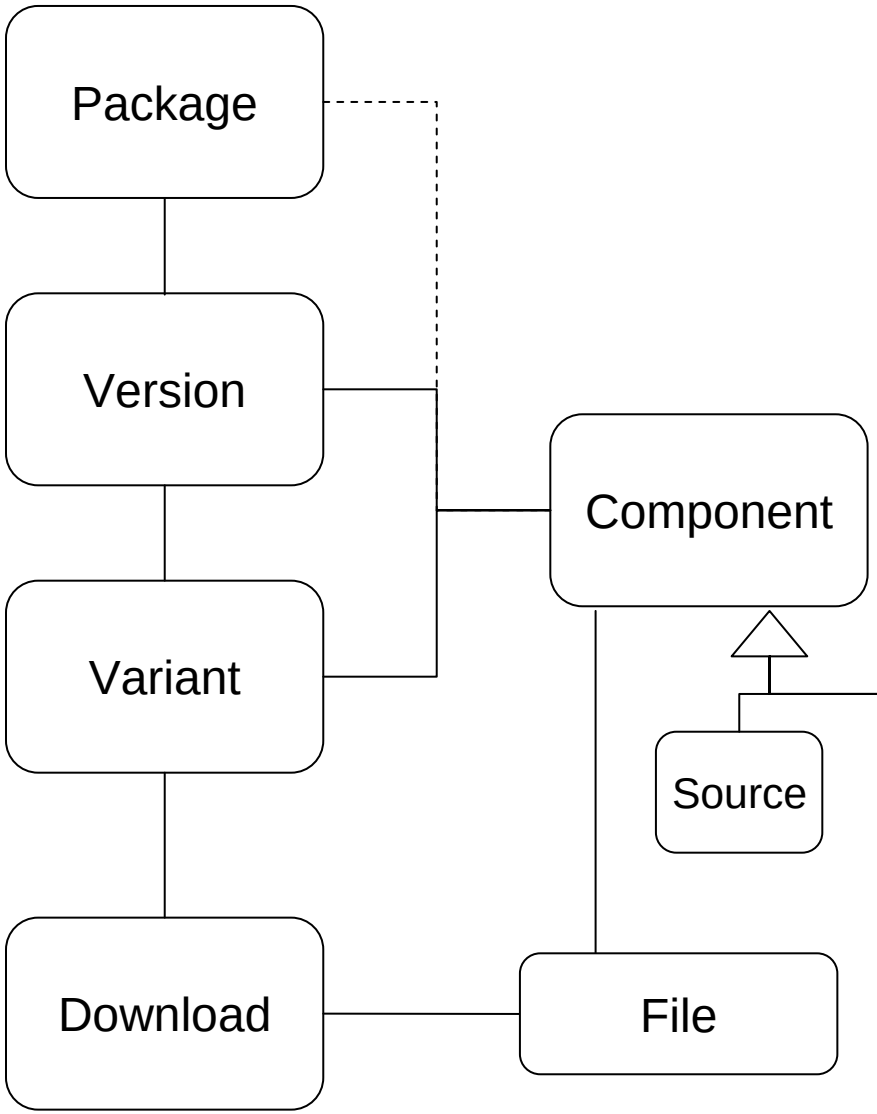


Provide a general model of software digital objects
Relate each concept in the model with a set of significant properties

For different preservation approach, we need different significant properties to achieve a desired level of performance.

- Package
  - The whole software object under consideration
  - Could be single library module, or very large system (e.g. Linux)
  - Comes under one “authority” (legal control)
  - Defines “gross functionality”
- Version
  - Releases of the system
  - Characterises by changes in detailed functionality
- Variant
  - Versions for a particular platform
  - Characterised by operating system and environment
- Download
  - A particular instance of a particular variant at a particular location
  - Ownership
  - An individual licence
  - Fixed to particular MAC or IP address, URLs etc.

Component Model

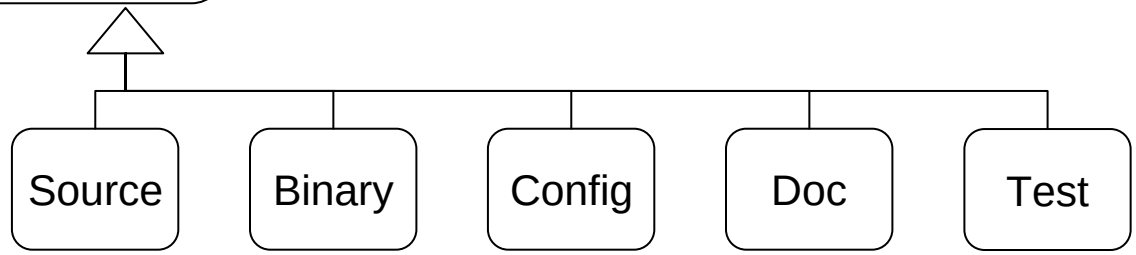


Each Version/variant is associated with a set of actual software components

Components have dependencies between them

Significant properties can be associated with components as well as on the version/variant

- the significant properties of a component may be of a different object type



Significant Properties

- Package Properties
  - Ownership and legal control, licencing
  - Provenance
  - Gross functionality:
    - Description of what the package does
    - Major input and outputs
    - Categorisation under a controlled vocabulary (e.g. GAMS)
  - Software architecture principles
- Version Properties
  - Source Code
  - Detailed functional description
    - Input formats, output formats, API, algorithm, error handling
  - Set of components and their dependencies
    - Including configuration and build as necessary
  - Programming languages
  - Usage documentation
  - Test cases
- Variant Properties
  - Precompiled binary
  - Specific operating system
  - Specific hardware platform if needed
    - Including any dependencies on peripherals
  - Specifics on machine performance
    - RAM and disk space, Processor speed, screen resolution
  - Compiler version
  - Dependent library or auxiliary tool version
  - Any variants on version components
  - Specific installation instructions for the variant
  - Documentation on any behavioural modifications
- Download Properties
  - Specific files
  - Specific environmental variables
  - Specific licencees, licence codes and conditions
  - Specific URLs or file paths
  - Specific MAC and IP addresses

Conclusions

- Limited experience out there of software preservation

- Straw-man conceptual model and significant properties
  - Needs more testing and evaluation
  - Needs extending the range of software types.
- More consideration of User Interaction Model
- Software engineering methods
  - Software Testing
  - Software version control (e.g. SVN)
  - Software Lifecycle
  - Managing software libraries
  - Software Reuse
  - Get the Software Engineers Involved.
- Preservation and archiving standards
  - OAIS
  - InSPECT

Good software preservation is good software engineering

Questions?

http://sigsoft.dcc.rl.ac.uk/twiki/bin/view

STFC and Digital Curation

- STFC E-Science Centre interest in the preservation of its science outputs
  - Publications – library systems
  - Data – output from facilities, Petabyte Data Store, Data Centres
  - Keep the results alive and available
- R&D Programme in Digital Curation
  - Partner in the UK Digital Curation Centre
  - Coordinator of the EC Project CASPAR
  - VSR, SCARP, Parse-Insight, ….
  - Case studies in our own data
  - Roll-out to facilities

Preservation Approaches

- Adequacy: How do we know we have captured enough?
  - Depends crucially on *Preservation Approach*

- **Technical Preservation. (techno-centric)**
  - Maintain the original software (binary), within the *original* operating environment.
  - Sometimes maintain the hardware as well
- **Emulation (data-centric).**
  - Re-creating the original operating environment by programming future platforms and operating systems to emulate the original environment,
  - so that software can be preserved in binary and run "as is".
  - E.g. British Library
- **Migration (process-centric).**
  - Transferring digital information to new platforms before the earlier one becomes obsolete.
  - Updating the software code to apply to a new software environment.
  - Reconfiguration and recompilation – “Porting”
  - An extreme version of migration may involve rewriting the original code from the specification.

- Different preservation approaches required different significant properties
  - Use a notion of *Performance to assess adequacy*
  - *Test case suites as tests of adequacy*

Package Properties

- Ownership and legal control, licencing
- Provenance
- Gross functionality:
  - Description of what the package does
  - Major input and outputs
  - Categorisation under a controlled vocabulary (e.g. GAMS)
- Software architecture principles

- E.g. Xerces
  - Provenance: Apache Software Foundation
  - Licencing: Apache Software Licence
  - Gross functionality: XML Parser
  - Architecture: Module to convert input text files into machine processable data structures.

Version Properties

- Detailed functional description
  - Input formats, output formats, API, algorithm, error handling
- Set of components and their dependencies
  - Including configuration and build as necessary
- Programming languages
- Usage documentation
- Test cases

- E.g. Xerces-C++ Version 2.8.0
  - Inputs: XML 1.0, XML 1.1, XML Namespaces, XML Schema 1.0
  - Outputs: DOM level 1 & 2, SAX 1 & 2
  - API: http://xerces.apache.org/xerces-c/apiDocs/index.html
  - Programming language: C++
  - Components: http://xerces.apache.org/xerces-c/download.cgi
  - Usage: Programming Guide http://xerces.apache.org/xerces-c/program.html
  - Tests : samples http://xerces.apache.org/xerces-c/samples.html

Variant Properties

- Specific operating system
- Specific hardware platform if needed
  - Including any dependencies on peripherals
- Specifics on machine performance
  - RAM and disk space, Processor speed, screen resolution
- Compiler version
- Dependent library or auxiliary tool version
- Any variants on version components
- Precompiled binary
- Specific installation instructions for the variant
- Documentation on any behavioural modifications

- E.g. Xerces-C++ Version 2.8.0 for Linux x86-64
  - Operating System: Linux x86-64
  - Specific installation instructions: http://xerces.apache.org/xerces-c/install.html#Unix
  - Binary: xerces-c_2_8_0-x86_64-linux-gcc_3_4.tar.gz
  - Compiler version: GCC 3.4.x or later
  - Dependent tools: GZIP, TAR, GNU Make (for source)

Download Properties

- Specific environmental variables
- Specific licencees, licence codes and conditions
- Specific URLs or file paths
- Specific MAC and IP addresses

- E.g. Xerces-C++ Version 2.8.0 for Linux x86-64 on a machine
  - XERCESROOT set to a specific path