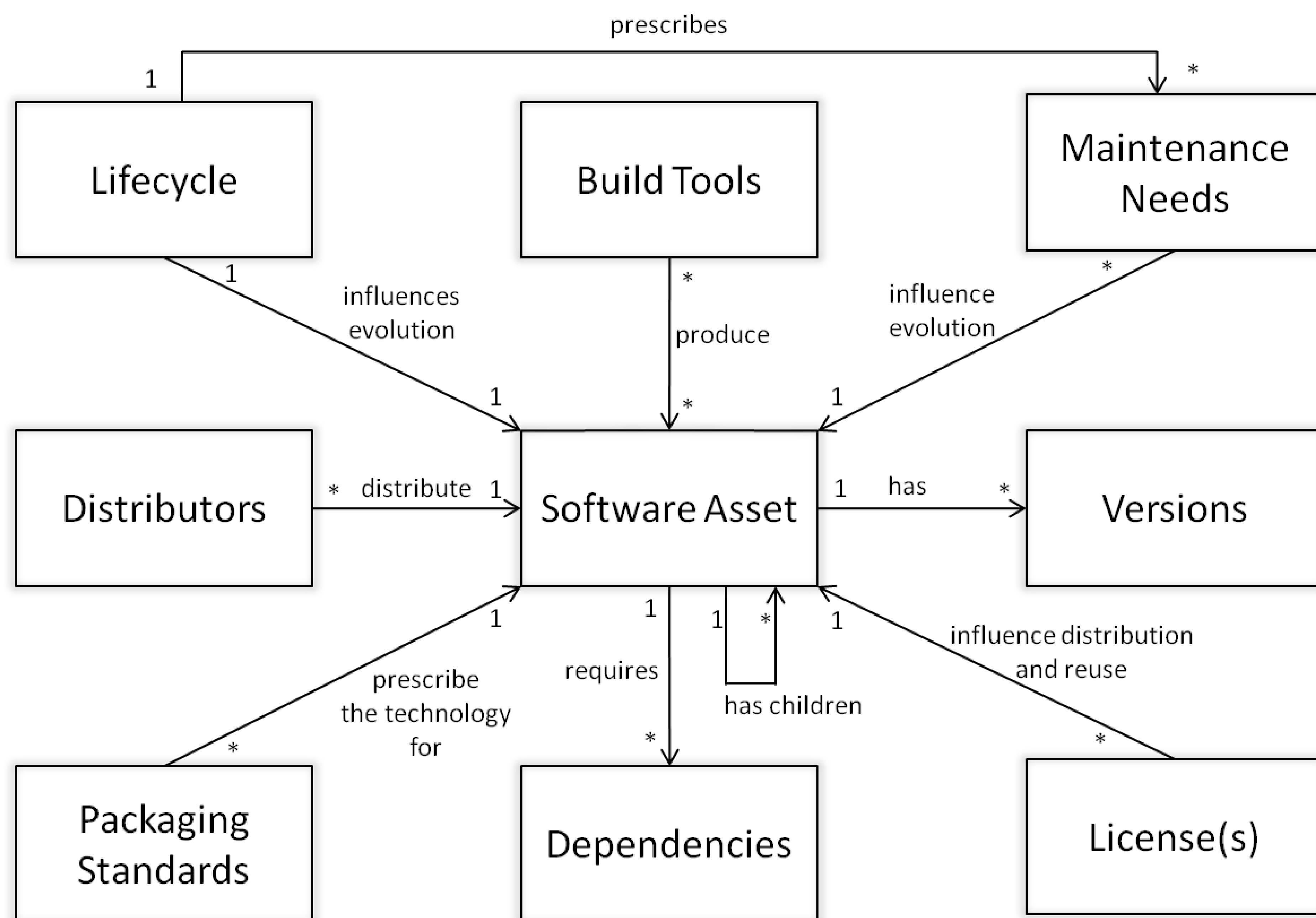


Packaging Software Assets for Reuse

Abstract

The reuse of existing software assets such as code, architecture, libraries, and modules in current software and systems development projects can provide many benefits, including reduced costs, in time and effort, and increased reliability. Many reusable assets are currently available in various online catalogs and repositories, usually broken down by disciplines such as programming language (Ibiblio for Maven/Java developers, PyPI for Python developers, CPAN for Perl developers, etc.). The way these assets are packaged for distribution can play a role in their reuse – an asset that is packaged simply and logically is typically easier to understand, install, and use, thereby increasing its reusability. A well-packaged asset has advantages in being more reusable and thus more likely to provide benefits through its reuse.

This presentation will discuss various aspects of software asset packaging and how they can affect the reusability of the assets. The characteristics of well-packaged software will be described. A software packaging domain model will be introduced, and some existing packaging approaches examined. An example case study of a Reuse Enablement System (RES), currently being created by near-term Earth science decadal survey missions, will provide information about the use of the domain model. Awareness of these factors will help software developers package their reusable assets so that they can provide the most benefits for software reuse.



A software packaging domain information model. Boxes represent concepts (attributes) relevant to the software package. Lines indicate a particular relationship – labels on the lines spell out the relationship type, and arrows indicate the directionality of the relationship. Cardinality of the relationship is indicated by a “1” indicating a 1-to-1 relation, or by a “*” indicating a 1-to-many relation.

Packaging Technology	Metadata Support	Ease of Use	Pervasiveness	Domain of Use	Open Source	Developed by CCSDS?	Compression Method
<i>Red hat Package Manager (RPM)</i>	Difficult to add metadata other than the existing package metadata defined by RPM.	Command line utility, extensible APIs, C interface.	Used throughout RedHat Linux systems, and available on many UNIX systems as well.	Software Package Installation.	Yes	No	Zlib
<i>Grid Packaging Tool (GPT)</i>	Any Metadata can be added with little effort. Metadata can be <i>per file</i> , or <i>per package</i> , or both.	Perl scripts and modules, extensible interfaces, command line tool, XML schema to describe software and tools..	Used throughout Grid community to install Grid software.	Grid Software Package Installation.	Yes	No	Zlib, XML
<i>Xml Formatted Data Units (XFDU)</i>	Any Metadata can be added with little effort. Metadata can be <i>per file</i> or <i>per package</i> , or both.	Java API, command line tool, XML Schemas to describe software and data model.	Emerging Standard, although recently used in DSMS prototype and at ESA.	Generic, meant to be used in any domain.	Yes??	Yes	XML, ZIP
<i>Space Formatted Data Units (SFDU)</i>	Supports user-defined metadata attributes, and a base set of metadata.	???	Used by NSSDC as a packaging method for creating AIPs	Space Data Systems.	Yes??	Yes	????
ZIP	Basic metadata such as file size, file modification date, etc. No support for adding metadata by default.	Java, C, C++, APIs, command line interface, extensible.	The de facto standard for compressing files.	Generic packaging technology.	Yes	No	ZIP
JAR	Basic zip metadata, along with META-INF directory describing package contents.	Command line interface, Java API interface.	The de facto standard for aggregating and compression binary java classes and packages.	Software Packaging.	Yes	No	ZIP
TAR	File Header Block,	Command line interface, API interfaces.	Long legacy in Unix, Linux, Windows; GNU project	File / Software Packaging	Yes (GNU)	No	ZIP
Mkpkg	Unix	Command line interface	HP's Unix	File / Software Packaging	Yes	No	?

A survey of different software packaging methods/technologies. We are in the process of expanding this table to include more technologies, covering more operating systems (the current survey is *nix-oriented) and technologies that are not open source.

Reuse Enablement System (RES)

How the RES packages assets it stores

- Asset metadata are captured, including name, description, creator, submitter, version, file size, platform, Reuse Readiness Level, and home page.
- User-created content such as average ratings and comments are also stored.
- The asset is either uploaded to the system as a package (e.g., a tarball), or a link is provided to where the asset can be downloaded.

How the RES is packaged for distribution

- A tarball (gzipped tar file) is created that contains the base XOOBS and added modules (with some modifications), an SQL dump of an empty and unconfigured database, an installation script, and documentation files.
- Currently, the installation script uses Perl and some non-standard Perl modules. We are currently examining how we can reduce these dependencies on items that are not required by the RES itself.

Two aspects of the Reuse Enablement System (RES) are summarized in the above case studies. The left side provides some basic information about how assets in the RES are stored/packaged, and the right side summarizes how the RES is packaged for distribution and use by others.

Authors:

Chris A. Mattmann**, NASA JPL / USC
 James J. Marshall*, INNOVIM / NASA GSFC (James.J.Marshall@nasa.gov)
 Robert R. Downs, CIESIN, Columbia University
 ** Presenter * Contact for WG information