MWSUG 2016 - Paper 36

Your Local Fire Engine Has an Apparatus Inventory Sheet and So Should Your Software: Automatically Generating Software Use and Reuse Libraries and Catalogs from Standardized SAS® Code

Troy Martin Hughes

ABSTRACT

Fire and rescue services are required to maintain inventory sheets that describe the specific tools, devices, and other equipment located on each emergency vehicle. From the location of fire extinguishers to the make, model, and location of power tools, inventory sheets ensure that firefighters and rescue personnel know exactly where to find equipment during an emergency, when restocking an apparatus, or when auditing an apparatus' inventory. At the department level, inventory sheets can also facilitate immediate identification of equipment in the event of a product recall or the need to upgrade to newer equipment. Software should be similarly monitored within a production environment, first and foremost to describe and organize code modules—often SAS® macros—so they can be discovered and located when needed. When code is reused throughout an organization, a reuse library and reuse catalog should be established that demonstrate where reuse occurs and to ensure that only the most recent, tested, validated version of code modules are reused. This text introduces SAS software that automatically parses a directory structure, parses all SAS program files therein (including SAS programs and SAS Enterprise Guide project files), and automatically builds reuse catalogs from standardized comments within code. Reuse libraries and reuse catalogs not only encourage code reuse but also facilitate backward compatibility when modules must be modified because all implementations of specific modules are identified and tracked.

INTRODUCTION

Software *reusability* is defined as the "degree to which an asset can be used in more than one system, or in building other assets."[i] In this text, *assets* represent only software programs or modules (such as SAS macros) that can be reused in subsequent software, but in a broader sense, *assets* can also refer to software documentation, risk templates, quality controls, and a host of other software-related items. Two knowledge management artifacts commonly implemented to document and manage assets and to facilitate reusability include the reuse library and the reuse catalog. Together, these tools facilitate documentation, organization, search, and retrieval of software so that developers can locate, understand, and implement code more effectively and efficiently. In many environments, these artifacts additionally include information about software risk so that SAS practitioners can understand the performance and relative quality of modules as well as the intended usage. Armed with this information, SAS practitioners can better assess whether code can be reused in its entirety, whether code must be repurposed to meet additional business needs, or whether code should be redesigned or developed from scratch.

Software reusability is thwarted when software is not adequately documented and organized, and unfortunately common in many software development environments, documenting code may be considered an afterthought rather than a necessity. In essence, software documentation is sometimes conceptualized as a distinct phase of the software development life cycle (SDLC) that follows software release, and which may juxtapose software operations and maintenance (O&M) activities. To the contrary, a widely held software development best practice is to infuse required documentation activities—however lean or fat they may be—into the software development phase, and possibly as early as software design. In doing so, whether operating in Waterfall or Agile development environments, this ensures that necessary documentation will be appropriately prioritized and provided sufficient resources (i.e., personnel) to be successful.

The automation of documentation can be a tremendous benefit to software development because developers can produce useful documents with little effort and, as demonstrated in this text, in some cases only through comments maintained within software. By standardizing the type and content of comments that appear in SAS code, software can be parsed automatically, all comments extracted and organized, and successful external documentation created through SAS reporting functionality. In addition to parsing comments, SAS functionality can also be parsed. For

example, the %MACRO statement always denotes the definition of a SAS macro, so the definition of all SAS macros can be collected, parsed, and organized with ease.

While numerous SAS white papers demonstrate both the successful standardization and parsing of comments within SAS program files to produce documentation automatically, a common obstacle within environments using SAS Enterprise Guide has been the disparity in file formats. SAS program files (e.g., those with the .SAS extension that are created by the SAS Display Manager) are ASCII text files that are readily ingested into SAS data sets for manipulation and analysis, whereas the SAS Enterprise Guide files (e.g., project files with the .EGP extension) are compressed, zipped files that must be extracted and interrogated before internal code can be extracted and analyzed. With the introduction of the ZIP access method within the FILENAME statement in SAS 9.4, this hurdle has been overcome, and all-SAS solutions can now iteratively read both program and project files (i.e., .SAS and .EGP files) to support automated document generation or other activities.

This text introduces the SCAVENGER macro that iteratively generates a list of all SAS program and project files within specified folders, extracts SAS programs from all project files, and parses all programs automatically to produce documentation that supports software organization and documentation. Once SCAVENGER has aggregated the collective information about a SAS environment into SAS data sets, separate software demonstrates how these data can be further parsed automatically to create reuse catalogs. With this pain-free documenting in place, SAS environments can dramatically improve their software reusability and reuse posture, further driving more efficient software development. And, because SCAVENGER itself is coded through modular software design that facilitates flexibility and reuse, other potential uses of SCAVENGER are also introduced and discussed.

REUSE LIBRARY

A *reuse library* is defined as "a classified collection of assets that allows searching, browsing, and extracting."[ii] Again, within this text, because *asset* references only software, a reuse library is nothing more than an organized software repository. Thus, reuse libraries can comprise a simple Windows directory structure on a shared network, implementation of the SAS Autocall Macro Facility, a SharePoint site, even more complex knowledge management software that provides software versioning, or any environment or tool along this continuum. To be effective, however, developers must be able to use a reuse library efficiently to research what software solutions have already been developed. When reuse libraries are poorly maintained, rogue practices develop and SAS practitioners are more likely to maintain individual rather than shared code bases because they cannot successfully locate software when needed.

In addition to promoting efficiency and effectiveness, reuse libraries also must be secure. Whenever developers are forced to check their code into a single, shared repository, the chance for corruption or overwriting increases, so SAS practitioners accustomed to developing in stovepipes will naturally be wary of ceding some control over their babies. To facilitate security of and faith in reuse libraries, shared code repositories typically espouse security measures such as version control or backups to guard against unwanted or untoward software modifications, and possibly user auditing and permissions to guard against unintended or malicious modifications. Only with shared code repositories can developers be sure they are using (or modifying) the most accurate, current, or complete version of specific software. Moreover, this level of coordination can make the software development process much more efficient, as a single code base can be tested, validated, and approved rather than wasting effort on testing various versions of similar software.

Reuse libraries are a critical first step toward organization of a collective software base for an environment, enabling basic search functionality to locate software modules. However, through the synthesis and analysis of collective software into a refined artifact, additional information and ease of use can be gained. Thus, reuse catalogs can be conceptualized as the metadata repositories that describe reuse libraries. For example, to determine the number of and ways in which a specific SAS macro is reutilized throughout an organization, a developer could conduct a global search of his network (i.e., the reuse library) to produce a disorganized array of information. Where reuse catalogs are incorporated, the developer would be able to access this information directly. Other benefits can include summary metadata such as program file cryptographic checksums, line counts, or versioning information that can be calculated automatically and added to reuse catalogs to promote greater understanding of and security in software modules.

REUSE CATALOG

A *reuse catalog* is defined as "a set of descriptions of assets with a reference or pointer to where the assets are actually."[iii] Reuse catalogs rely on the underlying structure and contents of reuse libraries but facilitate greater exploration and retrieval through additional metadata and information. Typical information found within a reuse catalog might include:

- software path and file name
- process, module, or task name (e.g., in SAS this is commonly a macro name)
- software description
- link to program file
- creation time-date stamp
- last update time-date stamp
- additional versioning information
- file size
- lines of code
- cryptographic checksum (used to validate program file integrity and constancy)
- risk (including known threats, software vulnerabilities, and technical debt)
- program prerequisites or dependencies (including other software that use/reuse the program)
- program inputs or outputs

The SCAVENGER program is capable of parsing most of these metadata and generates a user-friendly HTML report that promotes readability. This strategy benefits SAS practitioners who otherwise would be forced to sift through code manually and who thus would be less likely to reuse code already produced within their environment.

Moreover, because SCAVENGER iteratively parses all code within an infrastructure, subsequent reuses of software are cataloged. For example, the FINDVARS macro might be created and used to generate a space-delimited list of all variables found within a parameterized data set. Due to its flexibility and generalizability, FINDVARS would be an ideal candidate for inclusion into a reuse library, at which point other developers could begin to utilize in their respective software products. However, if FINDVARS needed to be modified—to increase functionality, improve performance, or reduce vulnerabilities—developers would have to check all individual uses of FINDVARS to ensure the proposed modifications were backward compatible to current usage and did not break software products using it. These types of audits are facilitated by reuse catalogs that depict all software using specific software modules.

Another tremendous advantage of reuse catalogs is their ability to drive the reuse-versus-redevelop decision, in which developers are often faced with the decision of whether to use (or cannibalize) existing code to build a software product or to redesign and redevelop from scratch. When inline comments within SAS programs include information about software capabilities, use cases, best practices, and vulnerabilities, developers are able to make more informed decisions about whether, how, and to what degree to reuse existing software modules in future software products. These metadata can be included within SAS comments and easily parsed by standardized commenting and automatic parsing thereof.

CONCLUSION

The SCAVENGER program expands the ability to interrogate SAS program files to include SAS Enterprise Guide project files and the program files that reside within them. With this capability, SCAVENGER iteratively and efficiently parses the collective body of shared SAS programs within an environment to produce a SAS data set that includes

metadata and information for all SAS programs. This text demonstrated one remarkable use of SCAVENGER—to create software reuse catalogs that can be used to facilitate and measure software reuse within an organization.

REFERENCES

[i] ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE)—System and software quality models. Geneva, Switzerland: International Organization for Standardization and Institute of Electrical and Electronics Engineers.

[ii] IEEE Std 1517-2010. *IEEE standard for information technology—System and software life cycle processes—Reuse processes*. Geneva, Switzerland: Institute of Electrical and Electronics Engineers.

[iii] Id.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Troy Martin Hughes
E-mail: troymartinhughes@gmail.com

APPENDIX A. SCAVENGER SOFTWARE

```
insert code here
```

APPENDIX B. REUSE CATALOG CODE

```
insert code here
```