Seven Tips for Building a Better Reuse Library

By Dr. Jeffrey S. Poulin
Systems Architect and author of Measuring Software Reuse, *published by Addison-Wesley*
13 July 2001

Installing a reuse library for your organization represents a significant commitment to improving your software and reducing application development costs. Once that your organization has the library, it should explore the most effective ways to maximize its investment in that tool. While you want to accept only useful components (see *Strategies for Implementing a Successful Reuse Library*), you also want to establish a level of quality components that keeps developers coming back for more. The challenge comes in balancing these objectives while keeping the library simple to use for everyone.

This article will explain seven steps that I have found essential to managing and growing a successful reuse library.

> **Reusable Components**: We tend to think of reusable components as software; in other words, things like functions, procedures, and classes. But reusable components can take many other forms. These include documentation, requirements, design and architectures, test data, test plans, and experience. What you choose to reuse depends on what has value to your organization. When I refer to a reusable component, I mean anything an organization finds reusable as well as the supporting information that goes with it. For example, a file containing reusable code may come packaged with many other files that contain helpful information such as hints on usage, test cases, integration instructions, and user documentation. All these things make up the reusable component [Poulin97].

Before you start: Assign a Librarian

The typical developer faces many technical and schedule challenges as part of the normal development cycle. With that in mind, organizations should make using the library as simple as possible. For example, we all know that a developer will rarely spend more than a few minutes trying to locate a component that he feels he could easily build himself. The same principle applies to populating the library: a developer will feel burdened by a submittal process that requires significant extra effort.

To streamline this process, every organization should assign a *Reuse Librarian* to handle the mechanics of submission. The librarian assists potential component contributors and takes responsibility for the "care and feeding" of the library. In addition to the more general administration and maintenance tasks, the librarian's job should include publicizing and even evangelizing the use of library. The remainder of the job involves instructing developers on techniques that will lead to higher reuse. Generally, this does

not require a full time commitment but the level of effort will depend on the size of the organization that the librarian supports.

Tips for Building a Better Reuse Library

Building a better reuse library requires a concerted effort by the organization to incorporate reuse into its software development process. Library administration and quality control fall into the domain of the librarian. Having one person responsible for the library ensures consistency, improves quality, and generally makes managing the library more efficient. The following list contains seven tips for populating a library that I have adapted from lessons learned while part of the IBM Reuse Technology Support Center (RTSC) in the early 1990's. Your librarian should consider these guidelines as part of a component submittal checklist:

1. Obtain the reusable component.
   The library tool normally includes a method for submitting components and designates a staging area on disk for submitting the components. Using this or similar method, the librarian must first receive the candidate reusable component (along with any supporting information) from a developer. The librarian may also create supporting information as he prepares the component for the library, such as documenting observations about usage or integration in a "readme" file. When the librarian completes this step, he has a collection of files that constitute the reusable component.

2. Approve the component for inclusion in the library.
   The librarian first must decide (often with the assistance of others) if the candidate component has a potential market of reusers. In other words, if the organization goes through the expense of managing this component for reuse, will anyone actually reuse it? Does the component represent a core function that developers repeatedly need when building applications? Don't waste time managing components that have little potential for reuse.

   An important validation step includes checking for any legal restrictions on the use of the component. Read the licenses of shareware and public domain software, as they often restrict including the software in products later offered for sale by the organization [Wu01].

3. Prepare the component.
   The librarian should initially prepare the component by performing basic quality checks. If the component includes code, then compile and run the code. Run the test cases provided and if necessary create test cases. Remember, "the first time a developer retrieves a component with a bug is the last time he will use your library."

   While preparing the components, look for consistency across the board, from the types of information provided, the level of detail, to the way the information looks. Run the code through a formatting tool. Consistency will encourage

developers to browse, locate, and reuse the components.

4. Load the components into the library.
   This should just constitute a few mechanical steps such as moving the files to appropriate directories, entering administrative data into the library tool, and making the component accessible to the library search engines. In some cases, the librarian may have to rebuild the library's search index to include the new component.

5. Check your work.
   Who said we didn't care about quality? The librarian must follow-up by performing a search for the new component to ensure that the search engine can locate the component. If the librarian can't find a component that he knows exists, then a developer probably can't find it either.

   Once the librarian verifies the search function, he should open each file to ensure that they did not get corrupted and that each displays properly using the required viewer (i.e., pdf files open with Adobe Acrobat). When the librarian completes this step, developers can start reusing the component.

6. Advertise!
   Send out an email announcement to the development team and make an pitch at the next department meeting. Explain and discuss the features, benefits and potential applications for the component. While someone may not need the component today, they might need it in the near future. Advertising reinforces the reuse culture. Publicizing successes and canvassing developers for their ideas will increase their buy-in to the program.

7. Maintain the component.
   Finally, the librarian supports the component as necessary by tracking problem reports (we hope not!), accepting enhancement requests, and answering general questions about its usage. Supporting the component includes keeping a log of who reuses the component so the librarian can update reusers, if necessary, about things like problems and upgrades. The usage log can also give very insightful information about what components people need and can help quantify the business benefits that the organization has received as a result of the library.

Conclusion

A reuse library requires an administrator to manage its growth, use, and promote its success. Reuse libraries succeed when they contain applicable, quality components that developers can easily find and use. This article gave seven tips for building a reuse library that meets these important goals.

References

[Poulin97] Poulin, Jeffrey S. Measuring Software Reuse: Principles, Practices, and Economic Models. Addison-Wesley (ISBN 0-201-63413-9), Reading, MA, 1997.

[Wu01] Wu, Ming-Wei and Ying-Dar Lin, “Open Source Software Development: An Overview,” *IEEE Computer*, Vol. 34, No. 6, June 2001, pp. 33-38.