Populating Software Repositories: Incentives and Domain-Specific Software

Jeffrey S. Poulin

Loral Federal Systems–Owego

Abstract

Reusable Software Libraries (RSLs) often form the core of an organizational reuse strategy. However, while RSLs provide a place to deposit software for use by others, RSLs do not guarantee reuse success. The implementation of a RSL depends on many factors including the availability of quality and useful software. Domain-specific considerations most often determine the usefulness of software and therefore should influence how an organization populates an RSL.

This paper presents IBM's[1] experiences with a corporate RSL, reuse incentive programs, and summarizes the results of an enterprise-wide initiative to develop reusable software for the RSL. The paper explains some of the issues surrounding a large RSL and defines a three-phased progression typical of corporate reuse libraries.

Keywords: software reuse, incentives, domain-specific reuse, software reusability, software reuse libraries

1.0 Overview

Recognizing the need and potential benefits of a coordinated reuse program, IBM formed the Reuse Technology Support Center (RTSC) for the purpose of promoting and forwarding "formal" reuse throughout the corporation. Although some sites had established reuse programs and most practiced some level of "informal" reuse, the RTSC sought to leverage the efforts of the corporation on a much larger scale. Doing this meant replacing the accidental, informal methods of locating, copying, and adapting software with a formal, systematic process of exchanging quality components that did not require modification before use. The RTSC worked with a designated reuse representative from each development site to bring reuse tools, processes, and software into the sites and to enhance the sharing of reusable assets between the sites.

Reuse on this corporatewide level requires, among other things, a source of reliable and applicable reusable software. To provide this source, the RTSC sponsored enhancements to an existing library system so that it could support and control the worldwide distribution of assets. The resulting RSL had features that enabled sharing of many kinds of components, including documentation, test cases, graphics, and of course, code. With a completed and fielded tool that provided the technical solution to highly distributed software reuse, the next task involved populating the tool with the differents kinds of components, or *parts*, to share.

Populating the RSL became a high priority; incentive programs started to encourage depositing parts into the RSL and using parts retrieved from the RSL. From the corporate perspective, however, progress seemed slow because the types and quantity of software available for general use quickly stabilized around a set of relatively low-level utilities and functions. To break this impasse the RTSC launched a corporate incentive program to encourage the creation and deposition of parts for the benefit of everyone. However, as this paper describes, this incentive program faced fundamental challenges in the attempting to satisfy the needs of a large, diverse corporation.

These needs come from the wide range of problem areas, or domains, in which the corporation operates. Unfortunately, this diversity limits the type and quantity of reusable parts that apply to more than one problem domain. This, in turn, limits the software that organizations can share or reuse. Experience shows that almost all reuse between organizations comes from a select collection of domain-independent software that usually does not exceed 15-20% of the total product. The majority of total reuse reported (sometimes up to 85%

[1] IBM sold their Federal Systems Company to the Loral Corporation on 1 March 1994.

of the product) comes from domain-specific libraries created by experts in their area.

IBM's many activities involve numerous different programming languages, different programming tools, and different areas of application. Every organization, and in many cases individual sites and projects, has dramatically different requirements for reusable parts. This paper presents IBM's experiences with a corporate RSL. It discusses reuse incentive programs in general and summarizes experiences with a companywide incentive program to develop reusable software for a corporate RSL. The experiences explain and confirm why some collections of software find uses in many organizations where other collections do not. The paper concludes that reuse organizations should make the most of domain knowledge and expertise. This approach relies on domain-specific parts centers where individual organizations and sites can best address their needs.

2.0 Reuse Library Considerations

Throughout the 1980's, many organizations based their reuse programs on a centrally managed reuse library; e.g., IBM, Siemens, Kodak, and numerous government agencies. However, although the library-centric metaphor guided early work in classification, storage systems, and other areas of reuse technology, the metaphor does not provide the best focus for setting up and running a reuse program. Quite often the *reuse program becomes the library.* When this happens, organizations spend their resources on library-related activities such as classification mechanisms and specialized tools rather than concentrating on inserting reuse methods and technology into the development groups that need it. Large corporations and organizations that have invested 80-130 person years of effort *each* on library-related activities have little to show for their investment other than reports that cite the large number of components contained in their respective libraries. While library-based reuse has resulted in modest levels of reuse, especially by providing a source of small, general-purpose components, it simply has not yielded a major change in the way most people develop software [6], [15], [19], [22].

Much of this results from the way organizations typically implement their RSLs. Although the theory behind creating a central software repository seems good, in practice, large RSLs suffer from a number of drawbacks that reduce their effectiveness. For example, RSLs normally contain few or no parts when they start out, making it hardly worthwhile to search them. The organization literally has a "library without books," and has difficulty convincing developers to even look in the library. An initial reaction to this dilemma calls for loading the RSL with whatever software the organization finds at hand. The organization might launch reusable parts "mining" expeditions, and might offer incentive programs to individuals depositing parts into the library. Libraries can fill quickly when an organization focuses exclusively on increasing the number of parts, but as the organization soon learns, developers will still not use the library because they know that it mostly contains software of questionable quality and heritage.

An alternative to indiscriminately filling the library lies in coupling the hunt for parts with requirements to comply with reuse quality or certification criteria. While this improves the quality and consistency of the parts in the library, it does not guarantee that anyone other than the original developer can use the part. The library may now contain well-tested, well-documented, and possibly supported software, but the organization soon realizes that developers do not use the library because they cannot find software that meets *their* needs. In summary, the large corporate reuse library tends to flow through the following three-phase progression:

The Reuse Library Progression

Phase 1. very few parts,
Phase 2. many parts of low or poor quality,
Phase 3. many parts of little or no use.

A further problem with large libraries lies with the overhead associated with the formality and rigor needed to manage large quantities of data. For example, large RSLs often use an extensive classification mechanism to describe software components. This mechanism provides detailed information upon which a user can search for and assess the usefulness of reusable parts. However, although (in theory) this information makes exact, detailed searches possible, quite often it only serves to confuse the user. First, the up-front presentation of large amounts of classifiers makes it difficult to quickly extract the key bits of information needed by the user to assess the component. Second, having a classification scheme requires users to receive training in its use; untrained users will not effectively use the mechanisms which the library carefully provides to assist them [26]. The existence of a standard method of classifying software may help solve this training problem and, in fact, a standards group has already

made progress in this area [32]. In the meantime, the lack of an industry standard method for software classification leaves RSL implementors to chose their own classification and retrieval methods. This means that users must not only understand software classification but especially must have an understanding of the scheme implemented by their own RSL [36].

3.0 A Corporate Reuse Library

The IBM strategic RSL provides a central repository for sharing, managing, and reusing software-related products to company sites worldwide. The repository runs on IBM's two major mainframe operating systems, Multiple Virtual Storage (MVS)[2] and Virtual Machine (VM)[2], to which nearly every member of the company has access. The RSL also complies with IBM standards for Systems Application Architecture (SAA)[2] and Common User Access (CUA).[2] A copy of the RSL runs at each development site and operates in cooperative fashion with the RSLs at all other sites. Together, the RSLs establish a system of shared libraries; any organization can establish a library to service the needs of a department, project, business area, or higher organization. The organization that creates a library owns, manages, and decides whether or not to grant access to RSL users from other sites or even other organizations within the site. Restricted access most often occurs during the initial phases of development because many groups feel reluctant to release software for general use before they have fully tested the code. Should the organization choose to never grant general access to their library, the RSL simply manages the software as a local reuse library. Otherwise, the tool supports distribution and control of the contents to the RSLs at other sites worldwide.

The RSL has the primary objective of providing a tool to enable reuse across a large, diverse enterprise. To provide the search mechanism for potential users to locate candidate components, the early design specified a detailed classification scheme based on the work of Prieto-Diaz and Freeman [29]. To locate a component for reuse, a user would invoke the RSL tool and select the library of components the user wanted to search. The user would then specify facet terms or values for the predefined classification attributes in the RSL. The tool would execute a search of the RSL database for components classified with values equal to those desired by the user. If the user felt satisfied with any of the components located by the search, the user could copy the component from the RSL onto the user's local disk space. The following table contains an extract of the classification scheme used in the RSL; a detailed discussion of the issues surrounding this kind of classification appears in [26]. The full classifier list used in the RSL reflects the size and diversity of the corporation that developed it; developers of everything from microcode to system test suites required specialized classifiers and terms that the scheme had to include.

Table 1. Partial listing of the IBM classifiers

Facet	**Description**	**Examples**
Algorithm	Technique to perform an action.	bubble, merge
Application Domain	Broad area of application.	advertising, aerospace
Certification Level	IBM quality rating.	Certified, as-is
Component Size	Component size in LOC or words.	512 LOC, 8245 words
Data Structure Characteristics	Implementation of the data object.	bounded, directed
Development Standards	Compliance with standards.	ISO 9000, DoD-2167A
Function	What the component does.	sort, add
Object	What the component acts on.	buffer, array
Implementation	Describes various techniques.	sequential, dynamic
Implementation Language	Programming language.	C++ Ada
National Language.	National language	English, French
Proven Hardware	Tested computer platforms.	RS/6000,[2] S/390[2]
Proven O.S.	Tested Operating Systems.	AIX,[2] OS/2[2]
Support Level	Guaranteed service provided.	Limited, Full

To enforce standards for component quality, the RSL also supports incremental stages of quality ranging from "use-at-your-own-risk" to "highly-trusted." The quality standards do more than simply indicate the reliability of

[2] Trademark of the International Business Machines Corporation.

a component, they seek to provide a way to enhance reusability. For a developer to efficiently use a software module, the developer should have access to other information such as design documents, integration instructions, test cases, and legal information. Table 2 contains a listing of the kinds of supporting information the RSL provides to potential reusers. A component receives one of three quality ratings (called a *Certification Level*) based, in part, on the completeness and quality of the information supporting the reusable module.

Table 2. Information helpful when reusing software

Attribute	**Description**
Abstract	Provides a clear, concise description of the component.
Change history	Describes changes to the code, who made them, the date of the changes, and why.
Dependencies	Describes prerequisite software and other software the component uses.
Design	Describes the internal design of the code and major design decisions.
Interfaces	Describes all inputs, outputs, operations, exceptions, and any other side effects visible to the reuser.
Legal	Provides a summary of legal information and restrictions, such as license and copyright information.
Performance	Describes the time requirements, space requirements, and any performance considerations of the algorithm.
Restrictions	Lists any situations that limit the usability of the component, such as nonstandard compiler options and known side effects.
Sample	Provides a usage scenario showing how the component applies to a specific problem.
Test	Contains information about the test history, procedures, results, and test cases.
Usage	Provides helpful information on how to integrate the component.

The RSL also provides component management functions to assist library administrators. These functions help librarians control library access, the distribution of components, and configuration management of the library and its contents. Although similar in many ways to the functions provided by source code control systems traditionally in use by development organizations, implementors of reuse libraries view these functions as specialized reuse requirements that a RSL must support [11]. These functions include:

- *Library access control.* Granting individuals or groups the ability to browse and extract components from a specific library.
- *Component user registration.* Tracking who extracts components for calculating library usage statistics and for problem notification, version control, etc.
- *Version control.* Maintaining multiple versions of the same component and notify reusers when new versions enter the RSL.
- *Problem reporting.* Routing a problem report to the original supplier and other users of the component if any reuser identifies an error, new requirement, etc.
- *Library shadowing.* Guaranteeing data integrity when copies of a library appear on different systems.
- *Standards enforcement.* Automatically enforcing reuse standards such as classification.

In summary, the RSL consists of a detailed, comprehensive reuse environment normally running as a standalone tool.

4.0 Reuse Incentive Programs

When the RTSC formed in early 1991, it faced a corporate RSL in the first of the three-phase Reuse Library Progression; in short, the RSL needed more parts. Although some portions of the company made heavy use of the library, user feedback indicated a need for a wider selection of quality and useful software. Some sites started incentive programs to encourage developers to both contribute to the RSL and to extract components for reuse. The next section will first discuss the dimensions of local incentive programs. Then, it presents the results of the corporate initiative.

4.1 Local incentive programs

Although the RTSC does not sponsor a centralized incentive program to encourage *individuals* to participate in reuse, it does offer guidance and encouragement to sites that want to start or have their own program. The guidance provided comes from the consolidated experience of approximately a dozen reuse incentive programs throughout the company and several others in industry; successful programs tend to have a number of common characteristics. Most importantly, the program should consider the *supplier-consumer* relationship described in

[3]. Therefore, the program should address the benefits derived from

- *Suppliers*, those who write and contribute quality reusable software and related information for use by others, and,
- *Reusers*, those who extract software from the RSL and incorporate the software in their products rather than develop it again. Reusers make up the "consumer" side of the reuse business relationship.

This section describes some of the key items to consider in creating a software reuse incentive program. Normally, recognition programs allot points based on supplying software for reuse, reusing software in a product, and on the level of a developer's participation in the program. An individual can receive points both from supplying and from reusing software. Accruing a certain total number of points (a plateau) results in the individual receiving prizes or financial awards.

4.1.1 Recognizing suppliers

Authors of reusable components bear an increased responsibility during development due to the added requirements, design, and testing they must perform. Furthermore, they should provide the additional information and classification required by the reuse practices of the organization. This information includes the

- extensive documentation,
- performance information,
- test cases and results, and,
- classifiers

outlined in Tables 1 and 2. When determining the proper recognition for authors, the program should address:

1. *Participation*

   Allocating points for first-time contributors helps motivate developers and generates interest.

2. *Popularity*

   Providing more recognition to contributors of very popular (that is, often-reused) components encourages contributors to look for high-reuse opportunities in their projects.

3. *Value*

   A large, generalized, and robust component provides more economic benefit to the organization than small, specific components.

A recognition program for suppliers should consider the three criteria above; the *Source Instructions Reused by Others (SIRBO)* metric [24] provides an excellent way to recognize part suppliers. To calculate the SIRBO for a supplier, multiply the size (in lines of code) of each component the individual contributes by the number of products or organizations actually using it. The supplier accrues points based on total SIRBO.

Using SIRBO helps ensure suppliers do not receive recognition for simply putting useless components into the library. It also rewards suppliers who identify widely needed functions, as proven by the number of uses their software sees. Finally, the metric rewards contribution of larger parts, the reuse of which normally have a higher economic return for the organization.[3]

4.1.2 Recognizing reusers

The decision to reuse ultimately results from a business analysis that shows a financial savings to the organization. The savings the organization receives forms the basis for individual recognition. Recognizing reusers also helps create a demand for reusable components that leads to financial benefits.

The benefits correlate directly to the amount of software consumed by the reuser. *Reused Source Instructions (RSI)* provides an excellent metric for rewarding reusers [24]. To calculate the RSI for a reuser, simply count the number of lines of code in each reused component. The reuser receives credit for RSI the first time a component appears in a new product; RSI do not increase from using the same component over and over. The recognition program normally assigns points based on the individual's total RSI.

[3] IBM uses SIRBO in its reuse productivity index, *Reuse Value Added (RVA)*, for the same reasons SIRBO makes a good metric for supplier incentives. Unlike other reuse productivity indices, RVA differentiates between actual use and the potential for use.

4.1.3 Recognizing teams

Another way to build supplier and reuser participation comes from having a team recognition award. A team incentive can reward a number of group activities, for example:

- Reward development teams who achieve either: (1) a pre-specified goal for the level of reuse on their project (for example, meet a Reuse% goal), or, (2) achieve an unforeseen, exceptionally high level of reuse.
- Reward "parts center" teams, e.g., those groups who primarily build reusable software for use by other teams. The number of actual uses of the software as reflected by the SIRBO metric indicates a strong customer focus and business return worthy of recognition.
- Reward any team with a high number of individuals participating in reuse activities. For example, provide each member of the team a bonus award when 100% of the team has received individual recognition points.

4.1.4 An example local incentive program

The following section illustrates how one development site implemented a reuse incentive program. The reuse coordinator at this site wanted to improve the implementation of reuse technology and develop a permanent motivator for both reusers and suppliers. To make the program simple and effective, the coordinator introduced a personal "reuse check" which looks very much like a stock certificate. As developers acquire "reuse points" as described below, they record the points on the reuse check. This process requires minimum administration and encourages developers to apply reuse over a long time.

Developers can acquire points for reusable parts in almost any form, to include program documentation, design specifications, test cases, and code. To count, the developer must:

- store the part in a publicly accessible repository, library, or tool,
- provide good documentation about how to use the part,
- name a person responsible for maintaining the part,
- have the part pass at least one inspection,
- have the part pass the Function Verification Test (a specific internal test process) if the part consists of code.

An approval board which consists of reuse representatives from projects across the site decide whether or not to accept parts submitted under the program. When the board approves a part and awards the points, the reuse coordinator records the points on the developer's reuse check. Points accrue as follows:

Points for Suppliers

Supplying a part - 1 point. In the case where a team jointly developed a part, up to four developers can each receive a point for supplying the part. A developer can receive up to 4 points per check for supplying parts. Providing points for simply supplying parts encourages participation in the program and intends to encourage developers to increase the general supply of software. Placing a maximum number on the points for supplying parts helps limit the tendency for developers who might only supply and not reuse; it also encourages developers to concentrate more on supplying quality, useful software rather than large quantities of software.

For each use of a part by another product - 2 points. The program does not limit how many points a developer can receive as a result of other products using a part. The SIRBO rule applies; the more successful the part, the more credit the supplier receives.

Points for Reusers

Reusing a part - 3 points. Reusers receive 3 points for the first use of a part in a new product. Using the same part multiple times in the same product does not count as reuse nor add to the point total. A developer can receive a maximum of 6 points for reusing the same part in two separate products.

When a developer accrues 12 points on the check the developer can redeem it for $1800. Although no developer redeemed a check in the first six months of the program, the reuse board approved approximately 50 points. The site coordinator expects to redeem several checks before the end of the first year of the program in 1994.

4.2 Incentive programs at other companies

Several companies have tried various incentives to encourage the supply and the consumption of reusable software [33]. Of the companies with reuse incentive programs most encourage reuse in one of two ways:

1. They appeal to the ego.
2. They appeal to the wallet or pocketbook.

However, although incentives can create a lot of reuse awareness at a very reasonable cost, in the long run they do not always change very much. Few companies attribute massive changes directly to incentives. In reality, the immediate needs of the project (e.g., schedule) take precedence over commitments for the additional engineering and development of reusable software. Because incentives provide an *external* and not *intrinsic* motivation, the desired behavior often stops soon after the incentives end. A site reuse coordinator from IBM Germany observed that the individual programmers that displayed the desired behavior (reused the most software) even before establishment of the incentive program mostly win when competing for reuse awards [35].

General Telephone Company (GTE) established their incentive program to help populate their initial library of components. In other words, they had a library in the first phase of the three-phase Reuse Library Progression. GTE encouraged suppliers of the RSL by offering informal awards of $200 for accepted contributions. Management also emphasized the importance of reuse during software development by setting a goal to achieve a reuse ratio of 20%. As the reuse program matured, GTE eventually increased this goal to 50% [29].

The Hartford Insurance Company found that informally soliciting submissions for their reuse library resulted in unsatisfactory software, something typical of a library in the second phase of the Reuse Library Progression. To improve the quality and usefulness of contributions, the company decided to explicitly contract with developers to build new components and subsequently maintain the components in its reuse library. Hartford Insurance Company recognizes employees who participate in reuse by giving coffee mugs and dinner-for-two awards. They also “appeal to the ego” by upgrading exceptional participants to more powerful hardware (for example, upgrading from a 386 to a 486-class machine), indicating managers’ “power tools for power programmers” philosophy [16].

Nippon Telephone & Telegraph (NTT) has a strong management emphasis on their reuse program. Development guides specify that projects must develop at least 5% of new code to meet NTT’s reuse standards and contribute the code to the NTT reuse library. The projects must also reuse at least 17% of the code in new programs by extracting the software from the NTT RSL. Nippon “appeals to the wallet” by giving informal awards of $200-$300 and a certificate to leading reusers [17].

American Telephone and Telegraph Bell Labs has some departments develop C++ libraries for use by the entire lab [5]. These libraries also contain C utilities from the UNIX operating system library. At one location, AT&T gives a “Thief of the Week” award (a plaque and dinner for two) to the programmer who reuses the most software from these libraries.

To provide an incentive to suppliers, Westinghouse took the “appeal to the wallet” approach [33]. To help populate its reuse library suppliers received $400 for contributions accepted to the RSL. As with GTE and NTT, Westinghouse stipulates that suppliers only receive incentives if the contributed software passes an acceptance standard established by the RSL. This helps address the problems of quality and usefulness experienced by libraries in the second and third phases of the Reuse Library Progression.

The United States Air Force also encourages reuse as part of the software acquisition process. Because the Air Force contracts most software development out to vendors, the Air Force encourages the contractors to use existing software and to deign new software for reuse. The policy states that Requests For Proposals (RFPs) must require contractors to propose software architectures that facilitate reuse and provide rewards to the extent that the contractor can shorten schedule and reduce costs by means of reusing software. The policy makes it easier to win a contract by giving source selection preference to contractors who include formal reuse in their proposals. It also provides special performance incentives, including award fees, and guarantees access to government-owned software to provide a reuse source for vendors [12].

Users of the Air Force Defense Software Repository System (AFDSRS) reuse library receive incentives for successfully reusing software extracted from the AFDSRS. The AFDSRS program office rewards their government users with a coffee mug and a t-shirt emblazoned with the AFDSRS logo. Recently they

have given away about 10 each t-shirts and mugs a month [1].

4.3 A corporate incentive program

While local reuse programs at IBM started to develop and mature, the overall quantity of parts available for general corporate use seemed to have stabilized. To resolve this situation, the RTSC decided that it should assume an active role in increasing the contents and use of the corporate RSL. To do so, the RTSC obtained corporate funding to sponsor the development of reusable software. However, moving from Phase 1 of the Reuse Library Progression meant trying to avoid the pitfalls of Phases 2 and 3. From the RTSC's experiences it recognized the quality issues surrounding the blind acceptance of parts. Avoiding the Phase 2 pitfall meant having a means to ensure parts deposited into the RSL met a suitable quality standard. Avoiding the Phase 3 pitfall meant ensuring that the parts deposited into the RSL would enjoy a wide area of application.

Faced with a modest amount of funding (less than a half million dollars), the RTSC needed to find a way to gain the greatest possible leverage in terms of delivered products and still meet the quality and usefulness criteria it felt important. The RTSC developed and launched the resulting program, called the "Parts Stimulation Program," early in 1992 to accomplish these goals.

The Parts Stimulation Program strategy centered on getting development organizations to realize that projects they currently had to complete would prove more valuable to themselves and to the corporation if they made portions of the software "reusable." To provide an incentive to organizations to make this determination the program funded development organizations to study their project and the business issues related to reuse [27]. To apply for the program development organizations submitted proposals describing:

- their application domain,
- the expected deliverables as a result of the funding, and
- an estimate of required funding.

If accepted, the Parts Stimulation Program funded the "front-end" expenses of conducting the reuse study, which included:

- performing a domain analysis [30],
- developing specifications for the reusable parts, and,
- completing a reuse business case detailing who else will benefit and the expected return on the investment.

Funding the front-end expenses proved an effective way to leverage a limited amount of funding. Because the organizations would retain responsibility for development costs, the program did not have to fund the most expensive portion of creating the parts. Finally, if the business case showed favorable returns the organization would agree to complete the work by:

- developing, testing, and deploying the parts,
- completing the parts at the highest IBM reuse quality level,
- loading the parts into the corporate RSL, and,
- maintaining the parts.

4.3.1 Incentive program results

Although the approach of funding reuse studies in existing projects greatly helped expand the scope of the program, the RTSC had immediate difficulty identifying candidate projects to develop software for use across the corporation. Out of 25 proposals submitted, only 3 had the potential to create software that might prove useful outside its specific domain. The remaining proposals would not contribute to RSL inventory according to the goals of the program.

Lacking suitable proposals to develop domain-independent software, the RTSC decided to fund alternative projects that, although did not result in software, might otherwise result in benefits to the corporation. These projects created reuse standards documents, developed tools, or studied (completed a domain analysis) an area for future software development. While these projects resulted in several useful deliverables, they did not contribute to the corporate reuse inventory.

The first two projects established reuse standards for the area of graphics, symbols, and illustrations and for the area of test cases. In each of these areas the RTSC did not yet have a well-defined method for packaging graphics or test cases in a way that other developers and testers could effectively find and use them. The software classifiers shown in Table 1 did not adequately describe these kinds of parts and the supporting information shown in Table 2 did not always exist for graphics and test cases. These projects established a standard reuse method in each of these two areas; deliverables included documents and a set of sample reuse products built according to the new standard.

The next set of projects produced tools that the RTSC felt would prove useful in the reuse process. One tool developed an APL language browser for the corporate RSL. This tool addressed the specialized needs of APL programmers who wanted to use the RSL to store and search for APL components. Another tool generated test cases and supporting documentation in accord with the new standard for reusable test cases. Likewise, information developers created a template-based tool to assist in the packaging of reusable parts for information development groups. The final tool helped develop and simulate graphical user interfaces in the interactive system productivity facility (ISPF).

The RTSC funded two projects which had substantial potential in the area of domain-specific reuse even though management would not immediately commit to finish the efforts. One project looked at the domain of communication services and the other looked at Application Programming Interfaces (APIs) for cooperative processing in a client-server environment. Both projects delivered a domain analysis and interface specifications for software in the domain.

Finally, three projects produced collections of general purpose software. One project developed a library of Presentation Manager[2] common services. Another project developed diagnostic utilities for C language programs. The last project built a set of communication class libraries for OS/2[2] and Advanced Peer-to-Peer Networking (APPN)[2] in the C++ language. Table 3 gives a summary of the projects funded by the Parts Stimulation program.

Table 3. Parts Stimulation Program Summary

Deliverable	**Number Funded**
Create Reuse Standards	2
Develop a Common Tool	3
Complete a Domain Analysis	2
Create Reusable Software	3
Total accepted=	10

5.0 Understanding What Makes Software Usable

Despite the RTSC's efforts, it discovered that the incentive program would not create software for corporate-wide reuse because of the limited kinds of software with such a broad scope. The experiences with the stimulation program forced the RTSC to focus on the fact that programmers cannot reuse a software asset unless they find it *usable* [33]. Programmers can only use a very small and select category of software in more than one problem area. To illustrate this, I group software into three categories based on range of *usefulness:*

1. **Domain-independent** software comprises the category of software with the widest range of usefulness. Domain-independent software provides the foundation for programming; e.g., abstract data types, container classes, graphical user interface functions, and math libraries. This software deals primarily with basic operating system functions or accessing prerequisite products such as databases.
2. **Domain-specific** software contributes the most to reuse but only *within the problem domain*. IBM's Operating System/2 (OS/2)[2] device drivers, navigational aids for aircraft, and financial services libraries serve as good examples of domain-specific software. Domain software concentrates on an area of an organization's business processes.
3. **Application-specific** software handles the specifics of a customer application; it consists of customized code needed to deliver a product. Although opportunities exist to reuse software within an application team, this software tends to deal exclusively with how one application can implement an application-unique function. This software has very limited reuse potential, *even within its own domain.*

Table 4 presents a concise summary of this model. It underscores that the state-of-the-practice in software reuse lies in constructing programs from building blocks of reusable components and that most of these components have a specific range of use.

Table 4. The three categories of software

	Class	Potential for Reuse	Examples
3	Application-Specific	Very limited, even within domain	Custom code written for an application.
2	Domain-specific	High, but only within domain	Avionics software, Financial functions
1	Domain-independent	Very high, even across domains	Abstract data types, GUI libraries, Math routines

5.1 Experiences with domain-independent software

The IBM RSL contains numerous (30+) domain-independent libraries for abstract data types (ADTs), graphical user interface (GUI), system functions, and operating system services. Individual software components in this category tend to consist of from 50 to 200 lines of source code, but those components typically contain only a single function or set of variables. Higher-level abstractions may include numerous interrelated functions or data that act together to perform a specific task, in which case the components may contain a significant amount of software. These components also tend to have very high quality; one well-used library consisting of about 80,000 lines of ADT routines has never had a single problem reported [13].

While domain-independent software enjoys a wide range of use, it only makes up so much of an application. Organizations and projects that draw nearly exclusively from the RSL's domain-independent software find that it rarely contributes more than 15-20% to their total product. For example, one organization which develops tools for developing, installing, and supporting information processing applications in the PL/I programming language experienced an average reuse level of 10% across its projects. Another organization which depends heavily on the RSL's domain-independent software reported annual reuse levels of 6% and 8% from 1992-1993. Table 5 lists further experiences with domain-independent software; the reuse levels reported comply with the counting model described in [25] and [27]. These results show limited levels of reuse from domain-independent software.

Table 5. Reuse levels with domain-independent software

Case	Product	Reuse%
1	Operating systems management	3.5%
2	Communications subsystem	20%
3	Communications subsystem	15.81%
4	Mainframe O.S. Product	8%
5	Mainframe O.S. Product	1.2%
6	Mainframe O.S. Product	2.3%

The results should not dissuade organizations from providing domain-independent software. Most organizations have already developed their own proprietary utility libraries or have licensed commercially-available products. This means oganizations can often distribute these functions and make the software available to their developers with minimal investment. I conclude that an easy, straight-forward, and low-cost way to get started in reuse consists of simply making it easy to build programs using domain-independent reusable components. Corporate-level groups can use this model and concentrate on supplying and supporting this software.

5.2 Experiences with domain-specific software

To increase the reuse levels shown above, experience shows that organizations must engage in domain-specific reuse. For example, as part of the Advanced Automation System (AAS) for the Federal Aviation Agency, the IBM Rockville site experienced reuse levels of 56% through the use of their avionics domain-specific software [20]. The AAS program team engineered and developed this common set of software early in the program and followed through with management directives regarding the use, configuration management, and version control of the shared functions.

The IBM Mid-Hudson Valley Programming Lab (MHVPL) experienced reuse levels of 43% in the Basic Control Program (BCP) of the MVS operating system through the use of their system-services macros and the use of Boeblingen Building Blocks [18]. MHVPL developed the shared system services after many years of experience with early mainframe operating systems and earlier versions of MVS. Developers recognized that many of the utilities in the source code control system consisted of specialized variations of similar functions. This led to a significant maintenance requirement and configuration control costs. Following an analysis of these utilities, they consolidated the functions into parameterized utilities which now form the

basis for many of the BCP components. These utilities have become highly trusted in quality and usefulness, resulting in lower maintenance costs, training costs, and more predictable software.

Loral Federal Systems–Owego conducted an extensive domain analysis as part of the Reusable Avionics Software Project (RASP) [8]. The avionics domain consisted of controls and displays for system monitoring, flight guidance, and navigation. The domain analysis identified reusable elements, reusable specifications, and configuration parameters for large reusable components. The result allowed automatic configuration of:

1. Requirements Specifications
2. Design Documentation
3. Ada Code
4. Test Documentation

and achieved reuse levels of 30% along with reduced development schedules. This work continues with the development of Domain-Specific Software Architectures (DSSAs) [9].

Other companies report reuse successes in a number of diverse domains. For example, AT&T has experienced reuse levels of 75-90% through the use of the *BaseWorX* applications platform. *BaseWorX* provides an architecture, a set of integrated software components, and numerous customer-support services for developing applications in the domain of network management and operations for the telecommunications market.

Recent feedback on the effectiveness of domain libraries in other companies shows similar results [15], [22]. Japanese corporations report very impressive reuse achievements with domain libraries consisting of only 70-100 assets [34]. Furthermore, successful experiences in a wide variety of domains such as telecommunications [7], avionics [10], distributed systems [14], and simulation/modeling [21] appear with regularity. Table 6 summarizes some results with domain-specific reuse; the reuse levels reported in the first three cases comply with the counting model described in [25] and [27].

Table 6. Reuse levels with domain-specific software

Case	Product	Reuse%
1	IBM Advanced Automation System	56%
2	IBM MHVPL MVS2 Operating System	43%
3	Loral Federal Systems Avionics	30%
4	AT&T *BaseWorX*	75-90%

Note that the existence of a domain library does not guarantee an organization will achieve high levels of reuse on a project. Some domains prove more conducive to high levels of reuse. The best results will come from a domain that:

- has limited scope,
- programmers understand,
- has matured to the point of relative stability, and
- requires a lot of custom development based on a common set of core functions [17].

If a domain has these attributes, creating a domain library generally happens in one of two ways; bottom-up and top-down. In the bottom-up approach the organization examines a domain after it has worked in the domain for a considerable time and has achieved a level of experience that allows it to identify domain-specific functions and relationships. The IBM Mid-Hudson Valley Programming Lab took this bottom-up approach using nearly 30 years of experience in developing mainframe operating systems. Loral Federal Systems–Owego also took a bottom-up approach when building its domain architecture for avionics, using over 40 years of experience in developing flight control systems.

In the top-down approach, however, the organization forms a team to study an area before it starts the main development effort. Organizations must use the top-down approach when they do not have the benefit of experience in the domain and normally employ a structured domain analysis method to help guide them through the process [2]. The resulting domain library continues to evolve during development but the initial work takes place early in the project. IBM Rockville took a top-down approach when it engineered and wrote their AAS shared libraries before the main AAS development effort began.

5.3 Organizing for Domain-Specific Reuse

The best place to develop domain libraries seems to lie with the people who have expertise in a given domain. Our most successful site reuse programs recognize this and designate a group (that IBM calls a “parts center”) to develop and maintain their domain library. Other companies have also started to take this approach [23].

This plan to create reusable software takes advantage of experiences which show that developing and managing domain-specific libraries seem to work best as a

decentralized activity. The "parts center" team typically consists of one to six programmers; in a larger group one person acts as a team leader [4]. The team reports directly to a high-level development manager and has responsibility for collecting requirements, designing, implementing and supporting (both with maintenance and technical consulting) all software shared by the groups they support. Furthermore, this "parts center" approach places the responsibility for developing domain-specific libraries with the experts in the domain. Although the domain experts will turn to the corporate group for supporting standards, education, and consulting, they have the knowledge to create domain-specific libraries that can best meet the needs of their environment and customers.

6.0 Conclusion

This paper examines the implementation of a corporate RSL to include its methods of classifying, certifying, and retrieving reusable components. It also discusses local and large-scale reuse incentive programs that aim to populate the RSL. Finally, the paper discusses the relationship between domains, the types of software that developers need within domains, and the types of software typically found in large RSLs.

Faced with a corporate RSL that it believed required an influx of reusable software, IBM launched a corporatewide incentive program to encourage domain-specific projects to develop and deposit domain-independent software into the library. Although the incentive method provided a very effective way to leverage a modest resource, the program attempted to develop software with a scope beyond what programmers could actually use worldwide.

Organizations should not exclusively focus their organizational reuse efforts on a central repository of reusable assets. Although this approach has enjoyed a lot of attention, it has largely failed. Programs that base their success on metrics such as "total lines-of-code in the RSL" and give monthly status reports on "total lines-of-code submitted to the RSL" do not understand the importance of domain-specific software and therefore the real usefulness of their RSL's contents.

The three-level model of software can provide a helpful guideline for defining responsibility for the development and maintenance of shared software. The corporate group can assume responsibility for domain-independent software and take the lead in providing standards, resolution of common issues, and technical consulting. This group then can assist local organizations to establish project and site "parts centers" to meet their own domain-specific requirements.

7.0 Acknowledgements

I would like to thank Will Tracz and Kyle Brown of Loral Federal Systems–Owego and the anonymous reviewers whose many helpful comments and suggestions contributed significantly to this paper. I derived some of the positions in this paper from the presentation given in [28].

8.0 Cited References

[1] Air Force Defense Software Repository System (AFDSRS). Contact the *AFDSRS Customer Assistance Office, Bldg 856, Room 265, Maxwell AFB, Gunter Annex, AL 36114.*

[2] Arango, Guillermo, "Domain Analysis Methods," in Software Reusability. Wilhem Shaefer, Ruben Prieto-Diaz, and Masao Matsumoto, eds. Ellis Horwood, Chichester, U.K.,1993.

[3] Barnes, B.H. and T.B. Bollinger, "Making Reuse Cost Effective," *IEEE Software,* Vol 8., No.1, Jan, 1991, pp. 13-24.

[4] Bauer, Dorothea, "A Reusable Parts Center," *IBM Systems Journal,* Vol. 32, No. 4, 1993, pp. 620-624.

[5] Beck, Roger, Satish Desai, Doris Ryan, Ronald Tower, Dennis Vroom, and Linda Mayer Wood, "Architectures for Large-Scale Reuse," *AT&T Technical Journal,* November/December 1992, pp. 34-45.

[6] Berg, Klaus, "CLASSLIB - Class Management and Reuse Support on a MVS Mainframe," *Reusability Track of the 1994 ACM Symposium on Applied Computing (SAC'94)* Phoenix, Arizona, 6-8 March 1994, pp. 53-58.

[7] Ciardi, Mauro, Pierpaolo Marangoni, Cinzia Sternini, "SIP Library Project: A New Methodology for Designing OSI Systems Implementations," *Eleventh International Conference on Computer Communication,* Genoa, Italy, 28 Sept-2 Oct. 1992, pp. 83-88.

[8] Coglianese, L. and Smith, R., "Core Avionics Domain Analysis," *Proceedings IEEE/AIAA 11th Digital Avionics Systems Conference*, Seattle, WA, 5-8 October 1992, pp. 143-148.

[9] Coglianese, L.H. and Szymanski, R., "DSSA ADAGE: An Environment for Architecture Based Avionics Development," *AGARD Conference Proceedings: Aerospace Software Engineering for Advanced Systems Architectures*, Paris, France 10-13 May 1993, pp. 321-328.

[10] Cohen, Sholom, "Process and Products for Software Reuse in Ada," *Proceedings TRI Ada'90*, Baltimore, MD, 3-7 December 1990, pp. 227-39.

[11] Donaldson, Cameron, "InQuisiX: An Electronic Catalog for Software Reuse," *SIGIR Forum*, Vol. 28, No. 1, Spring 1994, pp. 8-12.

[12] Druyun, Darleen A., "Software Reuse Incentive Policy," *Crosstalk: The Defense Software Engineering Report*, January, 1994, p. 5.

[13] Endres, Albert, "Lessons Learned in an Industrial Software Lab," *IEEE Software*, September, 1993, pp. 58-61.

[14] Gomaa, Hassan, "A Reuse-Oriented Approach for Structuring and Configuring Distribute Applications," *Software Engineering Journal*, Vol. 8, No. 2, March 1993, pp. 61-71.

[15] Griss, Martin, "Software Reuse: From Library to Factory," *IBM Systems Journal*, Vol. 32, No. 4, 1993, pp. 548-566.

[16] Incorvaia, Angelo J and Alan M. Davis, "Case Studies in Software Reuse," *Proceedings of the Fourteenth Annual International Computer Software and Applications Conference*, Chicago, IL, 31 October-2 November 1990, pp. 301-306

[17] Isoda, Sadahiro, "Experience Report on Software Reuse Project: Its Structure, Activities, and Statistical Results," *Proceedings of the International Conference on Software Engineering*, Melbourne, Australia, 11-15 May 1992, pp. 320-326.

[18] Lenz, Manfred, Hans Albrecht Schmid, and Peter F. Wolf, "Software Reuse Through Building Blocks," *IEEE Software*, Vol. 4, No. 7, July 1987, pp. 34-42.

[19] Lubars, Mitchell D. and Neil Iscoe, "Frameworks Versus Libraries: A Dichotomy of Reuse Strategies," *Proceedings of the 6th Annual Workshop on Software Reuse*, 2-4 November 1993, Owego, NY.

[20] Margano, Johan, and Lynn Lindsey, "Software Reuse in the Air Traffic Control Advanced Automation System," paper for the *Joint Symposia and Workshops: Improving the Software Process and Competitive Position*, 29 April-3 May 1991, Alexandria, VA.

[21] Miller, Lawrence and Alex Quilici, "A Knowledge-Based Approach to Encouraging Reuse of Simulation and Modeling Programs," *Proceedings of the 4th International Conference on Software Engineering and Knowledge Engineering*, Capri, Italy, 15-20 June 1992, pp. 158-163.

[22] Neumann, Valerie A. "Moving Beyond Library-based Reuse," *Proceedings of the 6th Annual Workshop on Software Reuse*, Owego, NY, 2-4 November 1993.

[23] Ning, Jim Q., "Module Interface Specification and Large-Grain Software Reuse," *6th Annual Workshop on Software Reuse*, Owego, NY, 2-4 November 1993.

[24] Poulin, Jeffrey S. and Joseph M. Caruso, "Determining the Value of a Corporate Reuse Program," *Proceedings of the IEEE Computer Society International Software Metrics Symposium*, Baltimore, MD, 21-22 May 1993, pp. 16-27.

[25] Poulin, Jeffrey S., "Issues in the Development and Application of Reuse Metrics in a Corporate Environment," *Fifth International Conference on Software Engineering and Knowledge Engineering*, San Francisco, CA, 16-18 June 1993, pp. 258-262.

[26] Poulin, Jeffrey S., and Kathryn P. Yglesias, "Experiences with a Faceted Classification Scheme in a Large Reusable Software Library (RSL)," *Seventeenth Annual International Computer Software and Applications Conference*, Phoenix, AZ, 3-5 November 1993, pp. 90-99.

[27] Poulin, Jeffrey S., Debera Hancock and Joseph M. Caruso, "The Business Case for Software Reuse," *IBM Systems Journal*, Vol. 32, No. 4., 1993, pp. 567-594.

[28] Poulin, Jeffrey S., "Balancing the Need for Large Corporate and Small Domain-Specific Reuse Libraries," *Reusability Track of the 1994 ACM Symposium on Applied Computing (SAC'94)* Phoenix, Arizona, 6-8 March 1994, pp. 88-93.

[29] Prieto-Diaz, Ruben and Peter Freeman, "Classifying Software for Reusability," *IEEE Software*, Vol. 4, No. 1, January 1987, pp. 6-16.

[30] Prieto-Diaz, Ruben, “Domain Analysis for Reusability,” *Proceedings of COMPSAC '87*, 1987, pp. 23-29.

[31] Prieto-Diaz, Ruben, “Implementing Faceted Classification for Software Reuse,” *Communications of the ACM*,, Vol. 34, No. 5, May 1991, pp. 88-97.

[32] RIG Technical Committee on Asset Exchange Interfaces, “A Basic Interoperability Data Model for Reuse Libraries (BIDM),” *Reuse Interoperability Group (RIG) Proposed Standard RPS-0001*, 1 April 1993.

[33] Tracz, Will, “Software Reuse Maxims,” *ACM Software Engineering Notes*, Vol. 13, No. 4, October 1988, pp. 28-31.

[34] Tracz, Will, “Software Reuse Technical Opportunities.,” *Proceedings of DARPA Software Technology Conference*, Los Angeles, CA, April 28-30, 1992.

[35] Wasmund, Michael, *panel position*, in Wentzel, Kevin, “Software Reuse, Facts and Myths,” *16th International Conference on Software Engineering (ICSE'16)*, Sorrento, Italy, 16-21 May 1994, pp. 266-273.

[36] Yglesias, Kathryn P., “Limitations of Certification Standards in Achieving Successful Parts Retrieval,” *Proceedings of the 5th International Workshop on Software Reuse*, Palo Alto, California, 26-29 October 1992.

9.0 Biography

Jeffrey S. Poulin (poulinj@lfs.loral.com) joined IBM’s Reuse Technology Support Center (RTSC), Poughkeepsie, New York, in 1991. His responsibilities on the RTSC included managing the corporate incentive program described in this paper. He currently works in at Loral Federal Systems–Owego as a Principal Investigator on a series of Open Systems Independent Research and Development (IR&D) projects. A Hertz Foundation Fellow, Dr. Poulin earned his Bachelors degree at the United States Military Academy at West Point, New York, and his Masters and Ph.D. degrees at Rensselaer Polytechnic Institute in Troy, New York.