THE COSTS RELATED TO MAKING SOFTWARE REUSABLE:

EXPERIENCE FROM A REAL PROJECT

W. K. Krutz CDP
K. Allen
D. P. Olivier CDP

INTERMETRICS, INC.
San Diego, CA

INTRODUCTION. In the past few years, a great deal of attention has been given to the topic of software reuse and how to both design new systems and make existing products of the software life cycle reusable [1,2,3]. Unfortunately, many of the previous discussions have not addressed the costs associated with, and the problems encountered in taking systems, in various states of maturity, and proceeding to extract various functions from them and making those functions reusable. This is of particular importance in the Department of Defense (DoD) community in which the software acquisition cycle is long, the associated costs are high, and the need for reliability is often quite exacting. In an effort to better control costs, reuse of existing systems is being stressed. Increased emphasis is being given to the use of libraries of government-owned off-the-shelf software (GOTS) life cycle components for both new development and post deployment support and its associated operations and maintenance (O&M) phase [4,5]. This paper describes the costs experienced in incorporating software components into an organized reusable software development environment (RSDE).

For the purposes of this discussion, a reusable software component (RSC) consists of all of the software life cycle products related to a particular functional entity deemed to be reusable. Such an entity could be at the traditional CSC or CSCI level or down at the procedure level. The RSC thus consists in turn of a collection of all of the products used to specify that entity such as, but not restricted to:

- descriptive text abstracts
- requirements
- computer-aided software engineering (CASE) tool diagrams
- procedure design language (PDL) statements
- source code
- object code
- compilation 'scripts'
- test suites (data and conditions)
- test drivers

and

- any other associated documentation thought to be pertinent.

Along with the challenges of preparing software life cycle products for reuse is the accompanying need to provide the prospective user of such products with a capability to see a demonstration of the candidate reusable life cycle product. This is a problem we have addressed in our current work and have collected some cost statistics associated with it.

PROJECT BACKGROUND. One of our government clients develops software for Navy command, control, communications and intelligence (C^3I) applications. At any particular point in time there are at least a half dozen teams of software engineers developing and testing C^3I applications for this client both at the client's site and at a variety of different geographic locations around the United States. Typical development consists of systems ranging from 20,000 to 200,000 lines of Ada code. Our client's objective in establishing an RSDE is to make stable RSCs available to a larger circle of developers and to provide the capability for developing systems according to an official policy known as evolutionary acquisition [6]. Candidate software components for the RSDE come from software which first and foremost is identified as belonging to the tactical C^3I application domain. Among specialized products

have parsers capable of processing select message types and tactical decision aids (TDAs), specialized software products that are used to aid the field commander in the making of immediate decisions relating to the location of his own or hostile forces.

We have developed an RSDE which is hosted on SUN Microsystems and derivative workstations, it consists of an integrated development environment supported by a sophisticated man-machine interface (MMI) in which a reusable software library (RSL) is a key part. In developing our RSL, we have developed a specialized series of Prieto-Diaz [7] facets describing our application domain as well as key words within each RSC which are a further refinement of our customer's needs. We've also provided a capability to demonstrate a number of the RSCs by providing software test 'harnesses' for them. This demonstration capability will figure in our subsequent discussion below.

Among the activities relating to the development of the RSDE included the production of a Reusable Style Guide (RSG) describing the nominally acceptable standards of documentation and source coding style required for RSC product sets.

After examining a number of graphical user interface (GUI) products, we designed and built the RSDE user interface making use of the TeleUSE(tm) GUI builder and the MOTIF environment [8] which is consistent with the standard operating environment (SOE) previously adopted by the customer's associated activities. This interface mandated certain standards be followed consistent with our RSG previously mentioned. Along with these standards comes the costs associated with assuring that each candidate RSC met these standards.

As we have worked with the ideas embodied in the RSG, they have led us quite naturally into the next phase of our work which has been to use our RSG as a basis in determining the potential reusability of some existing systems of immediate interest to our customer and to assess the costs associated with using our standards to achieve this goal.

TASKS ASSOCIATED WITH REUSE. We have found that much of the cost associated with reuse to be related not only to the structure of a software design and the software architecture produced from the design, but also quite literally, to the physical exertion associated with attempting to locate enough of the pertinent parts and then on the actual condition in which the candidate RSC products are found. In only a few cases that we have examined, did there exist a complete set of life cycle products available for review, nor did all of the products examined even remotely resemble the format required for inclusion in our RSL.

Several examples serve to illustrate this quite well. In the first case, through a search of a national repository it was found that there 'appeared' to exist a language translator for a commercially-based language to Ada. Upon browsing the narrative text, it was decided to request the source code. Several days later the source code appeared - on 3-1/2" PC-compatible diskettes. Since we had no machines in our offices which could process these, we transferred the data to 5-1/4" diskettes using another machine located off site. Once this was done the diskettes were then telecommunicated from one of our office PCs to a nearby SUN compatible workstation. Now began the work of trying to compile all of the modules. After overcoming several differences in Ada implementations, we encountered a compilation problem with a very large Ada variant record and related CASE statement constructs deeply nested. When we were unsuccessful in compiling this Ada source code, we returned to the narrative and made a series of telephone calls and written correspondences in order to locate personnel familiar(?) with this product. Some 30 days later, we received a call with an offer to assist. Only then did we find out that there was a 'missing' file. Several weeks later we learn that perhaps this file cannot be located at all!

In summary, this cost of reuse amounted to in man-hours as shown in the following table:

Example 1 - Cost of Reuse

(Man-Hrs)	
1.0	Electronicscanningofnationalrepositorylookingforcandidateproduct
0.25	Request Product
0.5	Transfer from 3.5 to 5-1/4" media
2.0	Electronically transfer to SUN Workstation
2.0	Compilation Efforts
4.5	Miscellaneous long distance telephone calls, written correspondences
10.25 Man-Hrs	Total (Thus Far) Across Several Months

This effort amounted to the man-hrs of two (2) professional software engineers for over one (1) day just to prepare a **product** to then determine if it is then reusable for use as a tool in making yet

another product reusable. This example illustrated some of the unanticipated costs of reuse. Two specific components of such costs are : (1) unanticipated obstacles; (2) time lost to productivity (the cost of 'opportunity lost')

In this case the software to be reused/rehosted had been in the condition we found it for a number of years (without the missing file) yet it remained in this state - with no one who had encountered the problem having any one at a repository add a caveat to the documentation. The other cost here was associated with the lost productivity associated with an RSC candidate we were planning to convert (and include in the RSL) using this conversion program. In the case of this reuse effort, the lost productivity was measured as some eight weeks during which we could have been using the conversion program (or determining that it did not suit our needs).

Another example serves to show yet a different set of problems - even when the software engineer has what he believes to be complete information. In this case the software desired for reuse was a message handling subsystem which operated as a part of a larger system in a MOTIF user interface environment. A software engineer with another organization had extracted the subsystem from the major system in which it is a part. Our customer had found a great deal of interest in other agencies for obtaining this subsystem's functionality, in its entirety. Our task was to make this subsystem reusable by first transforming it into a free-standing product, capable of being demonstrated as a part of the RSDE. Unlike the previous discussion, this product was an established subsystem, known to function correctly within its original host context and application domain. In this case we were seeking to make this subsystem reusable and demonstrably free-standing from the former and also quite possibly the latter. The man-hours associated with extracting this subsystem were as shown in the following table (and include a minimal amount of documentation to show how to install the product):

Example 2 - Cost of Reuse

(Man-Hrs)	
4.0	Time needed to get the extracted product 'up and running' (on the host network) upon receipt from extraction. This time included: • time spent copying files from engineer's directory • attempting to run the executable (as instructed by the original extraction engineer), getting an error message •tried to follow instructions provided by the engineer in a README file. • Overcame several problems relinking, primarily because link library had been moved and compressed by system staff which required their help to fix) • README instructions specified editing an executable file in order to set environment variable (UNIX operating system processes typically require these). • Tried various things in effort to get environmental variables set properly. Left it with executable unable to log on to the data base management system (DBMS) required.
0.5	Recontacted the extraction engineer, took notes on suggestions provided.
3.0	Rebuilt DBMS tables because they had been wiped out in a DBMS crash. Lengthy efforts to track problems through source code. Finally got it running after creating several additional environmental variables and copying config files into it based on the system administrator's and extraction engineer's recommendation.
2.0	Attempted to reinstall in another directory (ultimately unsuccessful), making entry in database for RSL, modifying installation instructions, (and writing this summary being read here). Copied all files to tape for installation on singular workstation at our offices.
3.0	Devoted time to removing dependency on liboss.a. This involved stubbing out the error logger module and modifying yet another system get_config routine. • Created a blank version of table "security" using information gleaned from the source code. This stops the error message "OSQL3 -943" from appearing, which enables us to use the subsystem as a free-standing demonstrable product
12.5	Total Man-Hours Spent across several weeks

A vital lesson is illustrated here: just because the product is made free-standing and demonstrable at the customer's site does not necessarily lead to the conclusion that it will operate in that manner at other sites. In this case, the unexpected problems experienced were those associated with the requirement that a UNIX environmental variable to be appropriately set and the removal of a dependency of a security file access. These items had been present in the customer's network environment and absent on the single workstation in our offices. Our offices represent a more realistic environment more closely akin to that of a potential end user since, typically, an end user will not necessarily have all of the external items present at the original host site.

What written documentation existed (with which this subsystem was accompanied) consisted on a User's Manual for a previous version of the software. Written using MicroSoft WORD™, they are currently only available for access on an Apple Macintosh™.

The documentation issue is a critical one and can add as much to the cost of reuse as any other process. This is because of dependencies that need to be documented often are not since the software was probably built without reuse in mind. When the software is made reusable, it is even more vital

that any dependencies be fully documented

As these examples have helped illustrate, one must consider a widely varying number of possible conditions when considering the costs associated with reuse. As we've shown, the costs we found to be (some totally unexpected, but typical of our efforts in the environment in which we are working) were:

Source Files Incompatibility. Individual source code files were not readily accessible from diskettes into the UNIX environment and had to be transferred from the diskettes to the workstation through the use of a UNIX 'DD' command which transferred the entire contents of each diskette as a single file.

Source File Reformatting. An Ada utility program had to be developed to search through source code and properly format it into discrete software units.

RSC Abstract Development. Despite the written documentation provided with the source code, that documentation was not organized in a manner commensurate with the extracted RSCs. Additional costs had to be budgeted to develop documentation consistent with the RSG for inclusion in the RSL.

CASE Diagram Translation. Although CASE diagrams were provided, they were unfortunately done using tools not compatible with the CASE tool suite selected for the RSDE. Although the diagrams could serve only as electronic drawings, it was decided to transfer them to the workstation environment. In order to rehost the diagrams, (originally created with and resident on an Apple Macintosh II™), it was necessary to convert their representation format from that of the .PICT format of the Macintosh to the raster format required for the workstation. This converted format was then communicated across an ETHERNET™ interface from the Macintosh to the workstation where the file resides as a standard UNIX file.

Loss of time value of result due to unexpected problems in contacting appropriate personnel, obtaining appropriate resources, etc.

Unexpected coupling brought on by the sophisticated dependencies created by the network environments, operating system process requirements and numerous commercial off-the-shelf (COTS) and GOTS support products required.

Now the more traditional work identified as a cost of reuse could begin. During this phase of our analysis, we encountered costs more akin to those traditionally thought of as a part of reuse of existing products:

Closure Costs. The magnitude of the closure required provides a great deal of information as to the potential reuse value of the RSC. In the case of source code candidates, too much code makes the source code candidate cumbersome for reuse; too small a candidate limits the reuse to a highly specialized function.

Facet Determination and Domain Analysis. Once closure was determined, the next step was to determine which of the facets we had developed could best be used to categorize the RSC. This required additional application domain analysis to determine, within the context of our RSL's application domain, how best to categorize this RSC. We also assigned several 'key words' to our RSL for this RSC as an additional refinement to the facets.

Document Research. Any of the documentation provided with the RSC candidates requires at least cursory examination in order to determine its value, as well as to gain sufficient knowledge of the software architecture to facilitate dismantling products into RSCs.

Documentation Creation. Creation of documentation, beginning with the RSC abstract, and moving on into CASE diagrams.

Configuration Management. Each RSC's associated library units must be carefully tracked from the time it initially enters the RSL on through the various versions that are created. Such tracking requires sufficient sophistication in order that if, for example, the source code implementation in a version changes, yet the level of the RSC documentation is still consistent, that only the baseline version of the documentation is maintained; not yet another copy of the same documentation.

Some of the optional costs that may be incurred

include the following.

RSC Testing. Significant costs are incurred if a test suite must be provided in order to assure the quality of any software products.

Environmental Demonstration Testing. Under some conditions it may prove advisable to provide an on-line test environment to permit the RSL end user with the opportunity to witness a demonstration of the RSC's capabilities.

THE NEED FOR STANDARDS. Our rehosting activities have quickly focused our attention on the need for standards for costing reuse of existing products in order to more accurately assess the costs related to our future efforts. We have found that the most efficient way to effect these standards, at this stage of our work, is to make use of a reuse cost estimation form in the form of a checklist as shown in the following figure.

LESSONS LEARNED. Software reuse, in short, (and in the words of Boris Belzer), 'takes bucks and guts' [9]. In a word, the barrier to reuse is financial and managerial. Reusable software doesn't get built if no one's willing to pay the price. In our work we have learned that: (1) many of the costs relating to making software reusable are those which need to be considered seriously and as 'worst case' scenarios, and not regarded as 'could never happen'; (2) our methods are workable but successful require persistence, detective work and not a little bit of good luck; (3) often the products you seek were never conceived with reuse in mind, they have been built by persons no longer available or with a vague recollection of the products; (4) as an RSDE and its RSL are developed they must be marketed in order to defer the costs of development and amortise them over a community of users; and (5) incentives for reuse must to be provided to encourage participation in your reuse 'consortium' and adherence to your standards for RSC. [Among ideas we've considered is providing a system of 'credits' (to be used for consulting services and available upon membership in the consortium), in exchange for use of the consortium's products and donation of reusable candidate products.]

FUTURE DIRECTIONS. We feel that we must extend our work into a larger amount of our customer's application domain and user community in order to determine if our results accurately reflect the costs typically encountered for software of the application domain we have been considering. We are also considering an independent research and development (IR&D) effort to automate some of the steps in the domain analysis process in order to decrease the associated costs [10].

Reuse Software Costing Checklist

	Media Format(s) Rcvd.
I. Electronic Format Products Received	
- Source Code	_____
- Command Files/Scripts	_____
- DOD-STD Documentation	_____
- CASE Diagrams	_____
- Object Code	_____
- Executables	_____
- Other	_____

II. Reuse Costs	Cost (Estmtd)	Actual [ManHrs.]
- Data Transfer	_____	
- Data Reformatting	_____	
- Document Review	_____	
- Abstract Preparation	_____	
- Facet and Key word Preparation	_____	
- Configuration Management	_____	
- RSC Testing	_____	
- RSC Environmental Demonstration Testing		
- Outside Resource Personnel Consulting(Systems Admin., DBMS,etc)	_____	
- OTHER [_ _ _ _ _ _ _ _ _ _ _ _ _ _]	_____	
Total Man-Hours	_____	

BIBLIOGRAPHY

[1] Arkwright, T. D.,"GLOBAL ISSUES IN REUSE FROM A REAL PROJECT", Ada Europe, 1986 Proceedings.

[2] Allen, K. "Software Reuse: Mining, Refining and Designing", Tri-Ada Proceedings, 1990, Baltimore, Maryland

[3] Krutz, W.K., "Software Reuse", seminar presented in four (4) European and Scandinavian cities during February-March, 1988, Technology Training Corporation (TTC)/State of the Art Limited (SAL)

[4] Krutz, W. K., "Post Deployment Support For Embedded Systems", Defense Computing, January, 1990.

[5] Krutz, W. K. "Post Deployment Computer Support", presentation for the Defense Systems Management College (DSMC) Management of Acquisition and Logistics Course (MALC), at USAF Space Division education center, Los Angeles, CA, December, 1988, May, 1990, April, 1991.

[6] SECNAVINST 5200 ACQUISITION OF SOFTWARE-INTENSIVE C^2 INFORMATION SYSTEMS, Department of the Navy, 5 January 1988.

[7] Prieto-Diaz, R., "Classifying Software For Reusability", IEEE SOFTWARE, January, 1987.

[8] Allen, K., "Comparison of User Interface Builder Products", INTERMETRICS Independent R&D Project, March, 1991.

[9] Belzer, Boris, Software Testing Techniques, 2nd edition, copyright 1990, Van Nostrand Rheinhold Publishers, p433.

[10] Barnes, B. and Bollinger, T. "Making Reuse Cost-Effective", IEEE SOFTWARE, January, 1991.

THE AUTHORS

William K. Krutz is a principal engineer and technical project manager with INTERMETRICS, INC., San Diego California. Currently Mr. Krutz is providing the technical direction for a project team developing a reusable software development environment for use in command, control, communications and intelligence (C3I) applications. Mr. Krutz has been involved in a wide variety of reuse activities in recent years including international speaking engagements in Europe and Scandinavia. Mr. Krutz holds a MASTERS degree in computer science from the Johns Hopkins University and is a member of SigAda, Society for Software Quality (SSQ) and the Data Processing Management Association (DPMA), as well as being a member of the adjunct faculty on the staff of a number of colleges and universities in Southern California.

Kent Allen has been involved with software reuse projects while working at Intermetrics, Inc. as a senior software engineer. Currently, he is developing the C^3I Reusable Software System (CRSS), a library of software components for Navy Command, Control, Communications, and Intelligence software development projects. This library system provides for the storage and retrieval of reusable software components for Navy C^3I systems, and allows engineers to access components through searches based on key words or Prieto-Diaz style facet value specifications. Components from the library are evaluated with the aid of demonstrations, online documentation, and CASE tool diagrams. Tasks for this project include prototype and application development, the development of standards for building reusable software components, and the definition of administrative and managerial procedures for making reuse a reality on actual, large-scale software development projects. Mr. Allen received his Master's degree in software engineering in 1988, from National University in San Diego. He has over eight years' experience developing software for government and scientific applications.

Daniel P. Olivier received an MS in Computer Systems Management from the Naval Postgraduate School in 1983 and a BS in Systems Engineering from the U.S. Naval Academy in 1977. He is an instructor of masters level project classes at National university establishing a reuse library of common Ada routines for class projects, software metrics, and program development support tools. He is currently implementing the C3I Reusable Software System (CRSS) Navy software library on Sun workstations using the Motif window manager. Mr. Olivier has made several presentations on software reuse to professional societies.