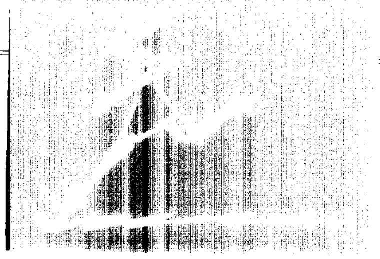


Large-Scale Industrial Reuse to Reduce Cost and Cycle Time

EMMANUEL HENRY AND BENOÎT FALLER, *Matra Cap Systemes*

◆ *For long-term reuse strategies to work, companies must realize short-term successes. This French company improved its time-to-market, productivity, and quality by pursuing reuse in two large industrial projects.*

Various strategies have been presented for an organization to improve software reuse in its engineering practices,[1] including the ones described in the box on p. 48. To increase reuse within a software company, most authors recommend significant changes in the organization's management, development methods, and tools.[2-4] All these changes — including the required psychological shifts — take time, delaying and sometimes wiping out the return on investment.[3,5]

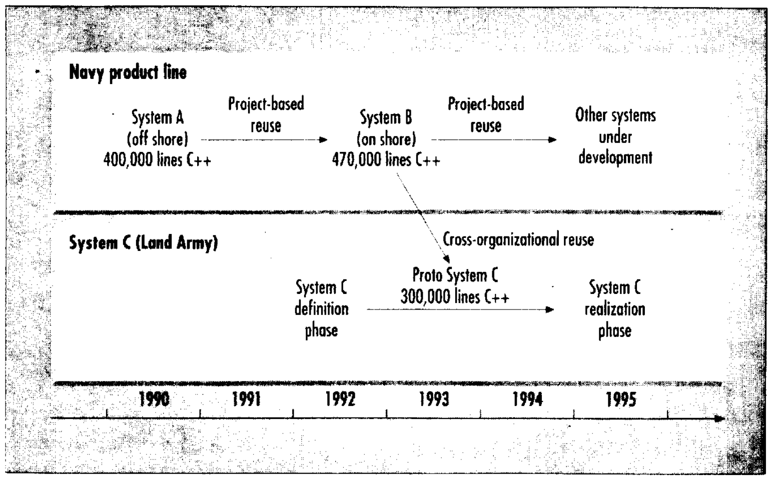
We think that such long-term strategies must be based on short-term reuse successes. In other words, from the beginning reuse must play a part in an organization's effort to improve quality and reduce cost and time-to-market. For this reason, Matra Cap Systemes, a joint venture of Matra Hachette and Cap Gemini Sogeti that develops information, communication, and imagery systems, founded its reuse strategy on pragmatic, opportunistic reuse-based projects. The box on p. 49 outlines the company's reuse plan, adopted in 1994.

In this article, we report the results from two large industrial projects, in which project-based and cross-organizational reuse improved time-to-market, productivity, and quality (as measured by error rates).

REUSE PROGRAM

One major reuse program is the development of Proto System C, a pro-

0740-7459/95/$04.00 © 1995 IEEE



Navy product line
System A (off shore) 400,000 lines C++
Project-based reuse
System B (on shore) 470,000 lines C++
Project-based reuse
Other systems under development
System C (Land Army)
Cross-organizational reuse
Proto System C 300,000 lines C++
System C definition phase
System C realization phase
1990
1991
1992
1993
1994
1995

Figure 1. *History of the three C³I projects. The size of these systems ranges from 300,000 to 500,000 lines of C++, excluding automatically generated human-computer-interface code. Systems A and B involved project-based reuse; Proto System C involved cross-organizational reuse.*

REUSE DEFINITIONS

There are several types of reuse, categorized either by whether they are planned or unplanned or by their scope and relationship to the organization:[1]

♦ *a posteriori*: To reuse existing assets that were not explicitly designed nor coded to be reusable. Also called *accidental*, *opportunistic*, or *unplanned* reuse.

♦ *a priori*: To design, code, and manage assets explicitly for them to be reused (involves a cycle of investment and a return on that investment). Also called *planned*, *anticipated* reuse.

♦ *project-based*: Reuse at the project level; project teams tend to perpetuate their original solutions when asked to develop new systems or products, leading to product families. It is usually a posteriori and involves the reuse of the system architecture, software architecture, and much of the design and code; the project organization, methods and development tools; and the competencies (people). Also called *product-line-oriented* reuse.

♦ *domain-based*: Reuse at the technical-domain level; focused teams implement very specialized technical know-how (algorithms, models, and so on) in reusable libraries, available for many systems or products. It is usually a priori and involves the reuse of small, independent assets, in which client projects ask the focused team for expertise, as well as for reusable code. Examples are telecommunication and security (cipher, key-management) libraries.

♦ *general-purpose*: Reuse at the organization level; project teams exchange existing solutions and assets as needed. It must deal with general-purpose assets such as basic structural classes, or common applications (text editors, system administration, and so on). It can be a posteriori or a priori and involves the reuse of small assets or sets of related assets. It is easier if it is associated with some transfer of people. Also called *cross-organizational* reuse.

REFERENCE

1 B. Bongard, B. Gronquist, and D. Ribot, "Impact of Reuse on Organizations," *Proc. Reuse '93*, IEEE Computer Society Press, Los Alamitos, Calif., 1993.

totype of an information system Matra Cap is building for the French Land Army. System C is a mobile system that will be used in the field by Land Army commanders. It displays tactical information on digitized maps and helps process orders and prepare and follow tactical land operations.

The prototype has been deployed on a limited basis into about 10 armored vehicles used in the field. Its life expectancy is short; its deployment is mainly for experimentation and to demonstrate the validity of the production System C, which will be deployed between 1998 and 2000. However, Proto System C is a fully operational system comprising about 300,000 lines of C++ code.

During the development of Proto System C, we reused significant assets (design, code, and tools) from System B, an information system we had developed for the French Navy. System B, a permanent system for the Navy headquarters on shore, manages and displays information relative to a naval region and communicates with French or Allied Naval Forces, plus several other systems. It mainly differs from System C in that it is a permanent (versus a mobile), regional, network-oriented (versus a data-processing) system, and because it is used by Navy top commanders (versus Land Army commanders).

System B was also a reuse project, derived from System A, an offshore system for Navy commanders embarked on big warships (like aircraft carriers). It manages and displays tactical information and helps users process orders and prepare and follow naval operations. Its scope is about the same as System C's, but for Navy Forces.

Figure 1 shows the history of these three projects, which are all command, control, communication, and information systems. The size of these C³I systems ranges from 300,000 to 500,000 lines of C++, excluding automatically generated human-computer-interface code.

Systems A and B involved project-based reuse; Proto System C involved cross-organizational reuse.

SYSTEM B REUSE

The Navy product line, of which Systems A and B are a part, consists of on- and offshore C^{3}I systems for the French Navy. These systems provide comprehensive computer-aided decision-support functions to the Navy's top commanders. They communicate among themselves and with third-party systems.

System A is an offshore system developed between 1990 and 1992. It was among the first significant projects developed in C++ by Matra Cap. Although delivery was delayed somewhat, System A was produced very quickly for software of its size (400,000 lines of code): Version 1 took 18 months (a three-and-a-half-month delay); version 2 was delivered three months later (a two-month delay). This was 30-percent faster than any comparable Matra Cap C^{3}I system at that time.[7]

System A is considered to be Matra Cap's standard for a C++ project. During the development of System A, Matra Cap won an open bid for the System B project, a C^{3}I onshore system for Navy headquarters. Two points made our bid especially attractive: Our technical credibility, which was established in our ongoing development of System A, and our ability to save the Navy money by massively reusing parts of System A. System B (470,000 lines of code) was delivered in 20 months.

Although System A was not built for reuse, Table 1 shows that the unplanned reuse in the development of System B was successful as far as costs and quality are concerned. System B also produced, on average, a 37-percent improvement in productivity and a 35-percent improvement in fault rates over System A development.[7]

These results may be surprising, since unplanned reuse has been

TABLE 1
EFFECT OF PROJECT-BASED REUSE ON PRODUCTIVITY AND QUALITY

	System B
Size (lines of code)	470,000
Amount of code reused from System A	35 %
Increase in the average number of lines of code per day, per person over System A development	37 %
Decrease in the ratio of major errors to total recorded incidents at time of delivery below System A development	**35 %**

REUSE PLAN

Matra Cap formally adopted a reuse strategy in 1994, based on pragmatic and opportunistic reuse-based projects, and a midterm action plan. The plan adoption was largely facilitated by System B development success and other similar experience.

The Matra Cap reuse plan outlined four objectives:

1. Integrate development with reuse in our methodology and procedures by
♦ supporting reuse-based projects,
♦ producing a "with reuse" development guide,
♦ introducing reuse considerations in project-management procedures and guides, and
♦ proposing organization adjustments that would make reuse easier.

2. Provide the company with access to a database about existing assets by
♦ developing and experimenting with the database (it uses the client-server tools of the World Wide Web on the company, private network),
♦ populating the database with existing assets (code modules of different granularity, ad hoc development tools, documents embedding important and reusable know-how, such as design, documents or study reports), and
♦ deploying the system (a few servers and many Mosaic-type clients) into the company.

3. Inform and convince people within the company of the value of reuse, using articles in the company newspaper, memos, conferences, demos, training sessions, and so on.

4. Prepare "for-reuse" (a priori reuse) efforts by
♦ preparing "for-reuse" business cases,
♦ supporting and following domain-analysis projects, and
♦ defining reusability criteria and "for-reuse" qualification procedures.

We had identified these objectives in 1993 from a self-evaluation of the company reuse maturity, using the Reuse Capability Model.[1] The reuse plan was implemented in 1994 by a dedicated team that was then dispatched in the beginning of 1995 to the operational divisions to create a reuse network.

REFERENCE

1. T. Davis, "The Reuse Capability Model: A Basis for Improving an Organization's Reuse Capability," *Reuse Adoption Guidebook*, version 01.00.03, Software Productivity Consortium Services Corp., Herndon, Va., 1992, pp. A3-A29.

Best Copy Available

described as cost-ineffective. However, several factors help account for our success:

♦ System A's object-oriented design — the use of modularity and information hiding — made its design and code easier to reuse.[8,9]

♦ The scope of System A and System B were, in general, very similar.

♦ System A's organization and work force were massively reused (almost half of the System B developers had worked on System A).

This is a typical project-based reuse experience. Matra Cap considers this its standard for "with reuse" projects.

PROTO SYSTEM C

Figure 2 shows a typical Proto System C installation. Each installation includes several Unix workstations connected by a local area network. Installations communicate with each other via a radio network. The workstations comprising each system, which are housed as a unit on each vehicle, communicate among themselves through a local area network, but the communication among vehicles is done through the radio network. As the figure illustrates, the Proto System C project was divided into three subprojects:

1. A data-processing subsystem (250,000 lines of C++ code).
2. A radio-communication server (50,000 lines of code).
3. Integration of the system into vehicles.

Most of the reuse took place in the data-processing subsystem, which is the focus of the rest of this article.

TABLE 2
COMPARISON OF PROTO SYSTEM C DATA-PROCESSING SUBSYSTEM TO SYSTEM B

Proto System C Functional Set	Similar	Modified	New
FS1, 47 functions	43%	28%	29%
FS2, 52 functions	15%	46%	39%
Other functional sets	0%	0%	100%

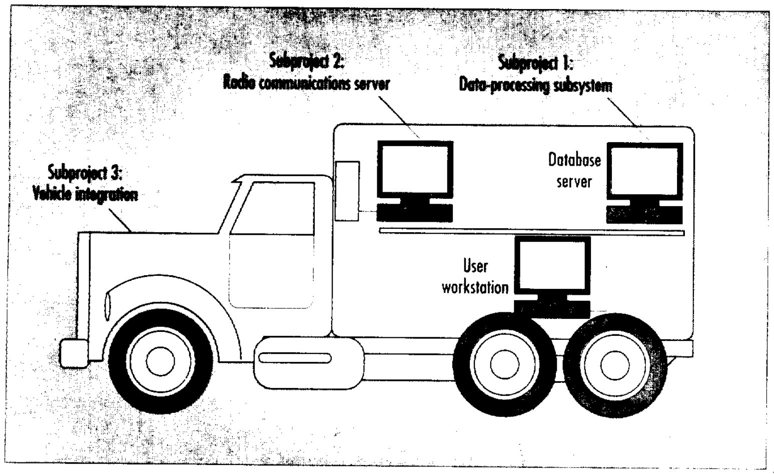


Figure 2. In a typical Proto System C installation, several Unix workstations are connected by a local area network. Installations communicate with each other via a radio network. The Proto System C project comprised a data-processing subsystem, radio-communication server, and integration of the system into vehicles. Most of the reuse took place in the data-processing subsystem

Data-processing subsystem architecture. Like System B's data-processing subsystem, Proto System C's is built on Unix and uses an X Window and OSF/Motif window manager, a 2D graphical library, a relational database-management system, and an Open Standard Interconnection message-handling system (X400 standard). It also uses commercial textual and graphical editors and spreadsheets.

The architecture for Proto System C's subsystem consists of seven functional sets that exchange information through a relational database:

♦ FS1: system exploitation (basic mechanisms and system administration).

♦ FS2: message processing (the treatment, preparation, and management of operational messages, carrying orders, reports, and so on).

♦ FS3 to FS5: specialized applications.

♦ FS6: situation management and display (tactical information onto a digitized map).

♦ FS7: interface to the radio-communication server.

Extent of reuse. During the specification phase, we realized that we could reuse only two of System B's functional sets, FS1 and FS2, as Table 2 shows:

♦ FS1 basic mechanisms, because

they were independent of the application domain;

♦ FS2 message processing, because message formats conform to a common military standard.

This reveals that the two systems are significantly different in scope. Furthermore, only 11 percent of the people (three of 27) who worked on this subsystem had worked on Systems A or B. These are characteristics of cross-organizational reuse.

As for design reuse, as we have mentioned we did reuse general concepts (the hardware and software architecture). Table 3 shows the extent of design-element reuse between System B and Proto System C's data-processing subsystems. (In our methodology, a design element is a large-grained object, implemented by five to 20 C++ classes.) This result was achieved thanks to good design and modularity of System B.

To measure code reuse, we conducted an analytical and mechanical comparison between System B and Proto System C using the Unix commands diff and awk. Table 4 shows the result: 34 percent of Proto System C's code was reused from System B. This figure is roughly the same as System A's code reuse in System B. However, the two reuse experiences were quite different in that System B reused most of System A's functional sets (project-based reuse), whereas Proto System C concentrated its reuse on two functional sets that formed the basic mechanisms used by the different applications (cross-organizational reuse).

TABLE 3
EXTENT OF DESIGN REUSE IN PROTO SYSTEM C (VERSION 1) DATA-PROCESSING SUBSYSTEM

Proto System C Functional Set	Reused	Modified	New
FS1, 39 design elements	51%	33%	16%
FS2, 25 design elements	28%	56%	16%
Other functional sets, 63 design elements	none	none	100%
Total, 127 design elements	21%	21%	58%

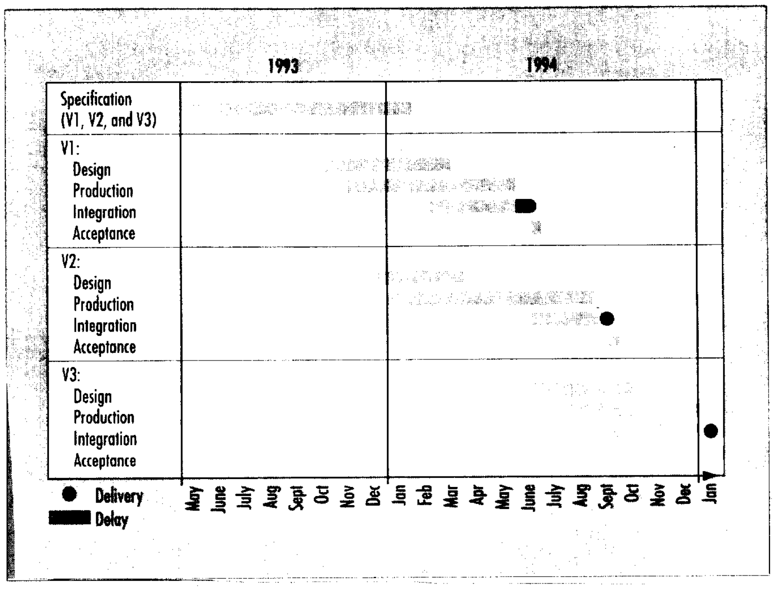


Figure 3. Development timeline for the three versions of Proto System C.

EFFECTS OF REUSE

To ready Proto System C for field tests during the last quarter of 1994, we were limited to three iterations. Figure 3 shows the development timeline for the three versions of Proto System C.

Version 1 consisted of the base software (FS1, FS2, and FS7) and two out of four applications (FS3 and FS6). Our goal was to quickly produce a minimal system to anticipate potential integration problems. This first version was delivered in June 1994.

Version 2 consisted of version 1, complements to FS3 and FS6, and application FS4. This version was delivered in September 1994, and was experimented on in the field by regular soldiers during a training exercise. The results largely validated System C concepts, and they led to some limited redefinitions of Proto System C version 3 user interface, but they were mainly used to finalize System C definition.

Version 3 consisted of version 2, complements to FS2, FS4, and FS6, and application FS5 — and thus filled all requirements. It was delivered in January 1995.

TABLE 4
EXTENT OF CODE REUSE IN PROTO SYSTEM C (VERSION 1) DATA-PROCESSING SUBSYSTEM

Proto System C Functional Set	Reused	Modified	New
FS1, 56,000 lines of C++	76%	3%	21%
FS2, 54,000 lines of C++	35%	8%	57%
Other functional sets, 70,000 lines of C++	none	none	100%
Total, 180,000 lines of C++	34%	3%	63%

TABLE 5
EFFECT ON AVERAGE NUMBER OF LINES OF CODE PRODUCED PER DAY PER PERSON

	FS1	FS2	Other Functional Sets	Total
System B versus System A				
Amount of code reuse	25%	35%	37%	35%
Productivity improvement	30%	65%	30%	35%
System C versus System A				
Amount of code reuse	76%	35%	none	34%
Productivity improvement	113%	26%	-7%*	18%

* *This figure is not shocking, because System A productivity was higher than average and because Proto System C better mastered C++, leading to more concise and effective code.*

TABLE 6
EFFECT ON RATIO OF MAJOR ERRORS TO TOTAL RECORDED FAULTS, AT TIME OF DELIVERY

	FS1	FS2	Other Functional Sets	Total
System B versus System A				
Amount of code reuse.	25%	35%	37%	35%
Change in faults rate	-27%	-27%	-36%	-37%
Proto System C versus System A				
Amount of code reuse,	76%	35%	none	34%
Change in faults rate	-17%	-47%	-31%	-33%

Reuse and cycle time. The effect of reuse on cycle time was especially significant for version 1, which was delivered in only 13 months — a 25-percent improvement compared to version 1 of System A.

Moreover, as Figure 3 shows, this 13-month cycle time is somewhat overstated because the initial nine-month specification phase was common to all three versions. In light of this, we consider version 1 specification to represent no more than six months of this initial phase. The corrected cycle time for version 1 is thus 10 months — a 44-percent improvement over System A version 1, which in itself showed a 30-percent faster delivery time than any comparable Matra Cap C^3I system at the time.

Figure 3 also shows integration phases of three to four months, a 25- to 50-percent improvement over System A versions, and almost no delivery delay (zero to 13 days), compared to System A's 56- to 107-day delay.

We achieved these short integration times because of the reused objects, which were in large part basic technical mechanisms that applications need to be initiated, to access data, to send and receive data, to communicate with other applications, and so on. Because they were reused, these mechanisms were readily available to applications developers, allowing them to concentrate on high-level functionality. Moreover, they could develop and test their applications in a preintegrated environment, which made system integration easier and faster.

Reuse and productivity. As Table 5 shows, the average productivity of developers on Proto System C was 18-percent higher than that of System A developers. However, productivity was lower on Proto System C than it was on System B. This increased expense resulted from the need to significantly adapt the function, design, and code of the FS2 message-processing function. This adaptation was anticipated during the specification phase: Table 2 points out that about 50 percent of FS2 functions were identified as reusable from System B, but with modifications. The need for adaptation was then confirmed during the design phase, as Table 4 shows.

Nevertheless, our cross-organizational reuse was extremely cost-effective on other parts of the system: We improved productivity on FS1 by 113 percent compared with System A.

Reuse and quality. Table 6 shows that the average fault rate during Proto System C development was 33-percent lower than System A — an improvement comparable to the 37-percent fault decrease in the development of System B compared with System A. Other projects report similar quality improvements due to software reuse, but do not specify if it deals with project-based or cross-organizational reuse.[10]

Although the overall improvement in quality and time-to-market and the reduction in cost were not as striking in the development of Proto System C as in the product-line reuse on System B, we did achieve significant improvements. These positive results can be attributed to both reuse and the iterative nature of the development process.

The Proto System C project confirmed the interest of cross-organizational reuse for Matra Cap. As a matter of fact, we record more and more projects reusing large parts of existing systems, although they come from different departments in the company. This process is now facilitated, since we produced a reuse guide that draws on our experience with Proto System C and similar systems.

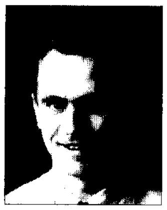
We now face another interesting consequence of our opportunistic reuse policy: several versions of similar modules, whose maintenance could be expensive if managed by different groups. For this reason, we are in the process of centralizing the maintenance of common code, like FS1 or FS2 functions. We believe that this new internal products policy will make a priori reuse easier in the long run. ◆

ACKNOWLEDGMENT

These projects were conducted under contract for the French Service Technique des Systèmes Navals (STSN) and the French Service d'Etude et de Fabrication des Télécommunications (SEFT).

REFERENCES

1. B. Bongard, B. Gronquist, and D. Ribot, "Impact of Reuse on Organizations," *Proc. Reuse '93*, IEEE Computer Society Press, Los Alamitos, Calif., 1993.
2. Y.A. Matsumoto, "A Software Factory: An Overall Approach to Software Production," *Tutorial: Software Reusability*, IEEE Computer Society Press, Los Alamitos, Calif., 1987.
3. M.A. Cusumano, *Japan's Software Factories - A Challenge to US Management*, Oxford University Press, Cary, N.C., 1991.
4. *Reuse Adoption Guidebook*, version 01.00.03, Software Productivity Consortium Services Corp., Herndon, Va., 1992.
5. B.H. Barnes and T.B. Bollinger, "Making Reuse Cost-Effective," *IEEE Software*, Jan. 1991, pp. 13-24.
6. T. Davis, "The Reuse Capability Model: A Basis for Improving an Organization's Reuse Capability," *Reuse Adoption Guidebook*, version 01.00.03, Software Productivity Consortium Services Corp., Herndon, Va., 1992, pp. A3-A29.
7. E.C. Henry, *The Impact of Reuse on Productivity and Quality in Software Development*, master's thesis, Massachusetts Institute of Technology, Cambridge, Mass., May 1993, pp. 41-76.
8. B. Meyer, "Reusability: The Case for Object-Oriented Design," *IEEE Software*, Mar. 1987, pp. 50-64.
9. D.L. Parnas, P.C. Clements, and D.M. Weiss, "Enhancing Reusability with Information Hiding," *Tutorial: Software Reusability*, IEEE Computer Society Press, Los Alamitos, Calif., 1987.
10. R.W. Selby, "Empirically Based Analysis of Failures in Software Systems," *IEEE Transactions on Reliability*, Oct. 1990.



Emmanuel Henry is the reuse manager in the Defense Systems Division of Matra Cap Systemes, Paris, where he has been in charge of the company's reuse-adoption program since 1993. He is also in charge of the company's private World Wide Web knowledge server. He has been a team leader in the C^{3}I Department of Cap Sesa Defense, Paris, where he was in charge of computer-security analysis, design, and implementation.

Henry received a BS in aeronautics and space engineering and an MS in automation from l'Ecole Nationale Superieure de l'Aeronautique et de l'Espace, Toulouse, and an MS in technology management from the Massachusetts Institute of Technology.

Benoît Faller is the manager of the Software-Engineering Department of Matra Cap Systemes' Research and Development Division, which deals with object orientation, distribution, prototyping, and reuse. He has been a project leader for Cap Sesa Defense, in the domain of war games and simulation systems. He has also worked as a researcher in artificial intelligence and object orientation at CNRS, the French National Scientific Research Center.

Faller received a PhD in computer science from l'Ecole Normale Superieure, Paris. He is a member of the French Association for Artificial Intelligence.

Address questions about this article to Henry at Matra Cap Systems, 6 Rue Dewoitine-BP14 Velizy Villacoublay, 78142 Cedex, France; henry@matra-ms2i.fr.