IMPACT OF ADA IN THE FLIGHT DYNAMICS DIVISION: EXCITEMENT AND FRUSTRATION

John Bailey
Software Metrics, Inc.
4345 High Ridge Rd.
Haymarket, VA 22069
703-385-8300

Sharon Waligora
Computer Sciences Corporation
10110 Aerospace Rd.
Lanham-Seabrook, MD 20706
301-794-1744

Mike Stark
NASA/Goddard
Software Engineering Branch
Greenbelt, MD 20771
301-286-5048

ABSTRACT

In 1985, NASA Goddard's Flight Dynamics Division (FDD) began investigating how the Ada language might apply to their software development projects. Although they began cautiously using Ada on only a few pilot projects, they expected that, if the Ada pilots showed promising results, they would fully transition their entire development organization from FORTRAN to Ada within 10 years. However, nearly 9 years later, the FDD still produces 80 percent of its software in FORTRAN, despite positive results on Ada projects. This paper reports preliminary results of an ongoing study, commissioned by the FDD, to quantify the impact of Ada in the FDD, to determine why Ada has not flourished, and to recommend future directions regarding Ada. Project trends in both languages are examined as are external factors and cultural issues that affected the infusion of this technology. This paper is the first public report on the Ada assessment study, which will conclude with a comprehensive final report in mid 1994.

INTRODUCTION

The Flight Dynamics Division (FDD) of the National Aeronautics and Space Administration's Goddard Space Flight Center (NASA Goddard) spends approximately $10M per year developing attitude ground support system (AGSS) and simulator software for scientific satellites. As the prime contractor in this area, Computer Sciences Corporation (CSC) develops most of this software.

Nine years ago, the FDD began investigating Ada and object-oriented design as a means to improve its products and reduce its development costs, with the intention of completely transitioning to an Ada development shop within 10 years. The FDD pursued Ada for reasons similar to those of other forward-looking software organizations in 1985. For example, Ada was considered to be more than just another programming language. Since it embodied several important software engineering principles and contained features to ensure good programming practices, its proper use was expected to lead to advances in the entire software development

process. Furthermore, through increased reuse, reliability, and visibility, using Ada was expected to reduce costs, shorten development cycles (project durations), and lead to better and more manageable software products. These goals of greater reuse, lower cost, shorter cycle times, and higher quality became the expressed objectives for learning and using the Ada language at the FDD.

The FDD piloted the use of Ada on its smaller, less risky projects (satellite simulators) that were regularly developed on a VAX minicomputer where a reliable Ada compiler and tool set were available. Early pilots showed promising results and led to a complete transition to Ada for developing simulators in the VAX environment. The results of the early Ada studies and successes have been presented at previous Software Engineering Workshops as well as at Ada- and OOD-related conferences (Reference 1). Unlike those previous papers, which compare the measures of Ada projects with the existing FDD baseline measures in 1985, this paper compares the Ada projects with contemporary FORTRAN projects. The impact of Ada is assessed in the context of the evolving FORTRAN process.

Table 1 presents the high-level characteristics of the Ada and FORTRAN projects included in this study. Notice that all Ada projects were developed on the VAX minicomputers, whereas all the FORTRAN projects were developed on IBM mainframes. Not only are the projects small or, at most, medium-sized by industry standards, but they tend to be short-lived, with operational lives ranging from just a few months to a few years. We have factored these organization-specific software characteristics into the recommendations made in the last section about the future use of Ada and FORTRAN at the FDD.

Table 1. Elements of the FDD Environment

Application	Computing Environment	Language	Typical System Size
Ground Support	IBM mainframes	FORTRAN	200 KSLOC*
Simulators	DEC VAX	Ada	60 KSLOC*

* Thousands of source lines of code

THE ADA EXPERIENCE

Over the past 9 years, the FDD has delivered approximately 1 million lines of Ada code. Figure 1 illustrates the growth of Ada experience in this environment. The curve shows the accumulated amount of code as each project was delivered (the time before the first project delivery is foreshortened for clarity). The scale on this figure is in thousands of physical lines of source code, or KSLOC (i.e., editor lines or carriage returns), and therefore includes comments and blank lines.

Although SLOC is the traditional measure of software size in the FDD, we used statement counts to measure software size for this study. We chose statement counts (i.e., the number of logical statements and declarations) because they are not sensitive to formatting and because

they are a more uniform indicator across the two languages both of functionality delivered and development effort expended (Reference 2). The average number of physical lines per statement varied somewhat over the period studied because of changes in programming style. By the last projects studied, the average number of lines per FORTRAN statement had risen from about 2 to more than 3, whereas the number of lines per Ada statement had fallen from over 6 to close to 4.

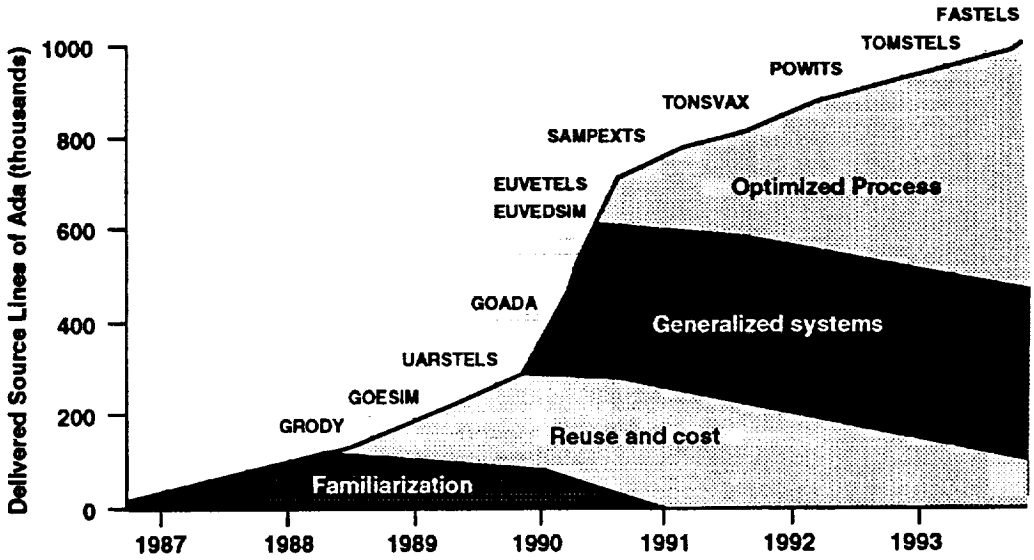


Figure 1. FDD Ada Experience and Focus

The four regions under the curve in Figure 1 give a rough approximation of the evolution of goals and objectives for the study and use of Ada in the FDD. Initially, the main concern was familiarization with the language, although the initial projects also stressed reusability as a major objective. Soon, the focus turned to the structured generalization of systems, and the success of these generalizations led to an overall improvement in the efficiency of the Ada software development process. Recently, there has been an additional focus on optimizing the development process specifically for use with the Ada language. This optimized process has been specified and documented in a recent supplement to the standard software development process guidebook used by the FDD (References 3 and 4). These Ada study goals for reuse, generalization, and process provided the framework for the evolution of the use of Ada in this environment. We discuss them further in the following section when we compare the Ada software with the FORTRAN software developed during the same 9-year period.

We examined the degree of usage of the many Ada language features on the various projects to verify that Ada developers were, in fact, using the full capability of this technology. We found evidence that the use of Ada has matured by looking at changes in the use of the language features over time. Figure 2 shows four views of the evolving language usage. Notice that the use of generics and strong typing increased, whereas the use of tasking decreased along with the average package size. Also, this maturation of language use appears to be leveling off, which suggests that sufficient thought has been given to using the language to enable a standard approach to evolve. These patterns indicate that the FDD developers have become skilled with Ada and have determined an appropriate style and usage for the language in this environment and application domain.

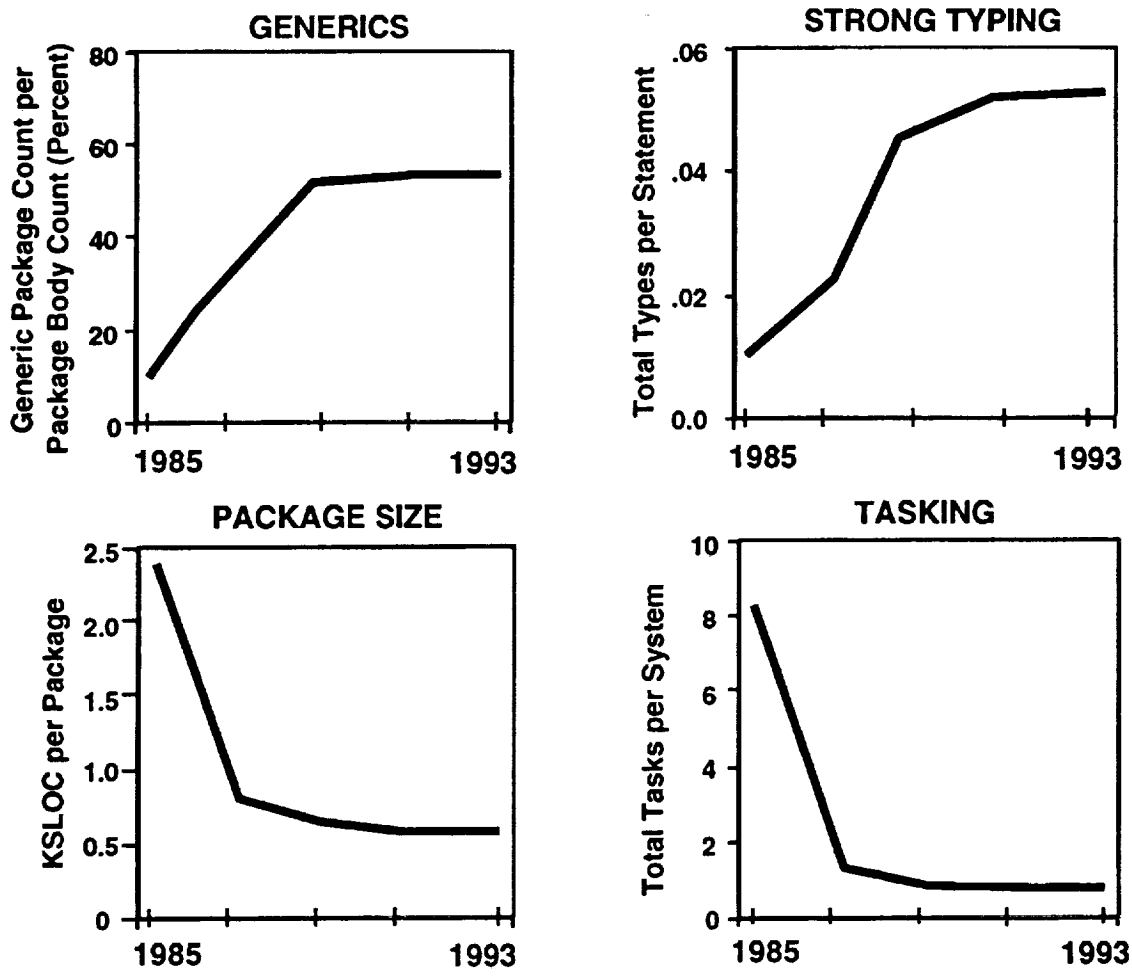


Figure 2. Maturing Use of Ada at the FDD

Comparing Ada and FORTRAN Baselines

The original FDD goals of increased reuse, lower cost (in terms of effort), shorter cycle times, and higher quality can be measured by comparing data from the Ada and FORTRAN projects. Previous papers (References 1 and 5) have documented improvement on Ada projects over the 1985 FORTRAN baseline. But, while the FDD was gradually maturing its use of Ada on the satellite simulators, the FORTRAN process also continued to evolve on the larger, mainframe projects. The following sections compare the evolving Ada and FORTRAN baselines between 1985 and 1993 in each of the initial four goal areas and in terms of the evolving software process. In this way, the improvements seen on Ada projects can be assessed within the context of the evolving FORTRAN baseline.

Reuse

During the 9 years that Ada has been used at the FDD, we have seen considerable improvement in the ability to reuse previously developed software on new projects. Figures 3 and 4 show, for Ada and FORTRAN projects respectively, the percentage of each project that was reused

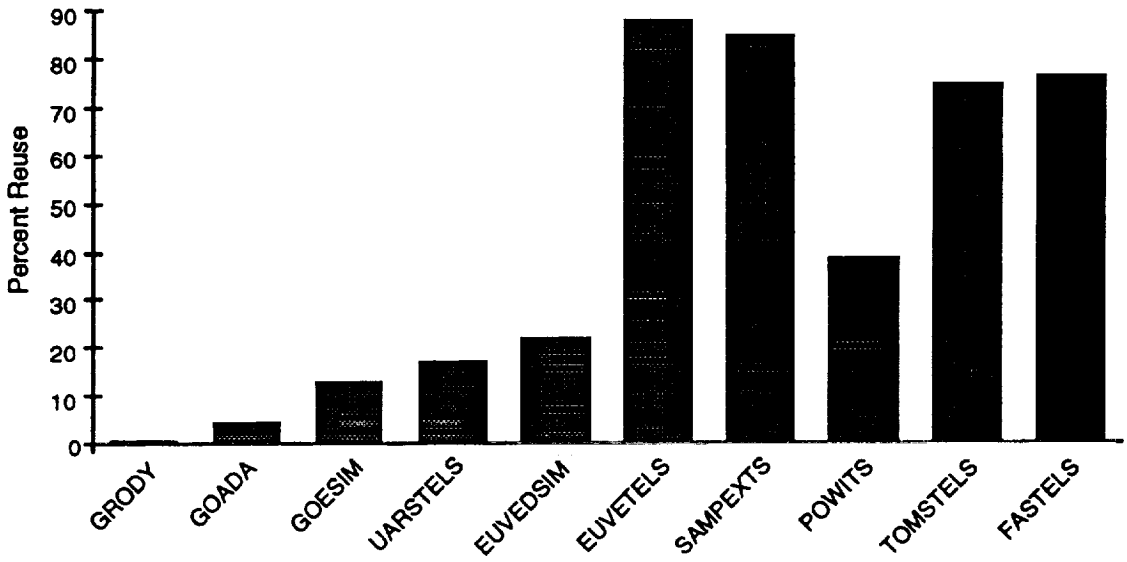


Figure 3. Verbatim Ada Reuse by Project

without change from previous projects. (The minimum unit of reuse is a single compilation unit; no credit is given if only a portion of a compilation unit is reused. The percentages are computed by dividing the total size of the reused compilation units by the total size of the project.) The first breakthrough in high verbatim reuse of Ada occurred in 1989 when a set of generics purposely designed for reuse were demonstrated to be sufficient to construct nearly 90 percent of a new project in the same domain.

The dip in the amount of reuse on the eighth Ada project was caused by a change in the domain that required modification to the Ada generics and additional new code development. Specifically, the original domain where high reuse was achieved was simulation software for

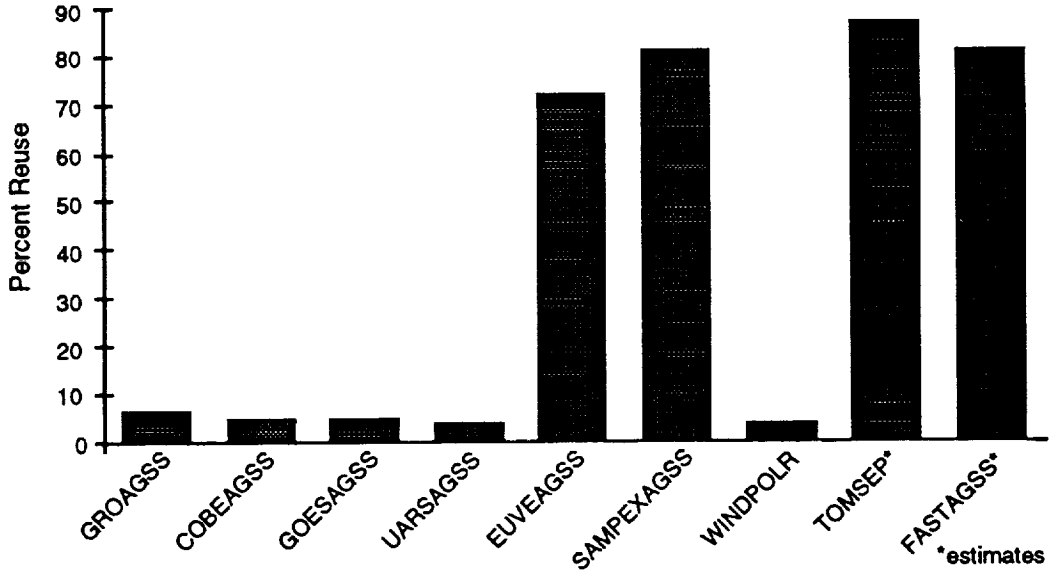


Figure 4. Verbatim FORTRAN Reuse by Project

three-axis stabilized spacecraft. When a spin-stabilized spacecraft was simulated for the first time, a substantial drop occurred in the verbatim reusability of the library generics. This incompatibility was rectified over time so that the generics can now accommodate either a three-axis or a spin-stabilized spacecraft. The slight drop in the most recent examples of reuse to around 80 percent, as compared with the earlier successes with high reuse that were closer to 90 percent, was caused by performance tuning on the latest projects. Performance issues are discussed again in the next major section of this paper.

Figure 4 shows the corresponding picture of verbatim reuse on the FORTRAN projects during the same 9-year period. At its peak, the amount of verbatim reuse achieved was nearly as great as with the reusable Ada generics, and the first successes occurred at nearly the same time as the first highly successful Ada reuse. (The first high-reuse FORTRAN project was the corresponding ground support system for the same satellite mission as the first high-reuse Ada simulator.) Again, a change in domain to spin-stabilized missions caused a drop back to the low levels of reuse observed on the earlier projects in the late 1980s.

The FORTRAN reuse approach differs from that used on the Ada projects, however. Instead of populating a reuse library with a set of generics that can be instantiated with mission-specific parameters, as was done in the Ada projects, the FORTRAN reuse library actually consists of two separate program libraries, together comprising nearly 400,000 source lines of code. One of these libraries is for three-axis stabilized spacecraft and the other is for spin-stabilized spacecraft. Subsystems from the appropriate library must be used in an all-or-nothing fashion. This style of reuse explains why there was a sharper drop in reuse when the change of domains occurred in the FORTRAN projects as compared with the Ada projects. In the Ada projects, it was still possible to reuse a sizable portion of the three-axis library without modification for the first spin-stabilized mission, whereas none of the large FORTRAN library to handle three-axis missions could be reused verbatim for a spinning satellite. The developers again achieved high levels of reuse in their FORTRAN projects by developing a separate complete subsystem library for spin-stabilized spacecraft that was analogous to the three-axis library. Since these two FORTRAN libraries embody over 80 percent of the functionality for any new ground support system, the FDD has since set up a special dedicated team to maintain them. This team is charged with keeping the software up to date with new requirements, while retaining its backward compatibility with previous systems. Estimates of the project-specific effort expended by this team are added to each FORTRAN project that reuses subsystems from the libraries.

An important distinction between the reuse styles adopted for the two languages is that the two FORTRAN libraries must be augmented as needed to handle new missions in their respective domains, whereas the Ada generics form a collective set of smaller components that requires little or no further modification to handle missions in either domain. To avoid the risk of introducing errors for existing clients, the maintainers augment the FORTRAN subsystems as necessary by adding new code rather than by generalizing or modifying their existing code. This causes the FORTRAN libraries to grow over time. In contrast, the Ada developers directly handle the generics needed for each project and further generalize them if necessary. By copying the generics into each project library that needs them, slight changes can also be made to eliminate unnecessary dependencies. The maintenance and configuration control disadvantages of having separate copies of the reusable components in each client project's

library are less an issue with the simulators, which have short operational phases, than they would be with the longer-lived AGSS projects.

Cost Reduction

Figures 3 and 4 clearly show the points when dramatic improvements in reuse were achieved in both the Ada and the FORTRAN projects. The first Ada simulator and the first FORTRAN ground system to exhibit high reuse were both written to support the Extreme Ultraviolet Explorer (EUVE) satellite mission. Because of the nature of satellite mission support, the simulator is typically completed first so that it can be used to test the ground system. (In the case of EUVE, the Ada simulator was completed about 4 months ahead of the corresponding FORTRAN ground system.) Since these first successes with reuse almost coincided, and since they are associated with measurable changes in the development approach, we divided the Ada and the FORTRAN project sets into two groups depending on whether or not they were completed before the EUVE experience.

The left-hand side of Figure 5 shows the average costs in hours to deliver a statement of Ada, both before the successes in reuse and since (EUVE and subsequent projects). The figure shows that the productivity of delivering Ada software has doubled since high reuse has been achieved.

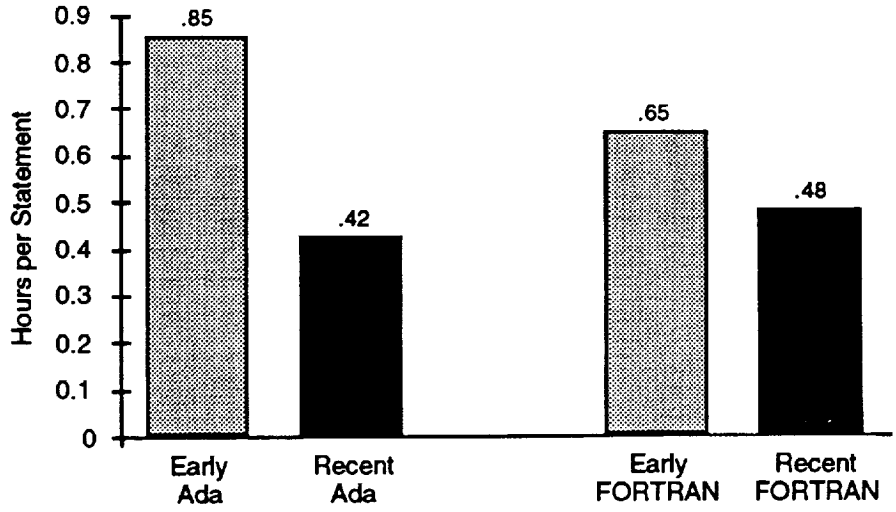


Figure 5. Effort To Deliver One Statement: Ada vs. FORTRAN, Early vs. Recent

The right-hand side of the figure shows the average costs in hours to deliver a statement of FORTRAN before and after the high-reuse process. Again, there is a marked improvement, though not as great a reduction as in the Ada projects. Although we have no hard data to normalize statement counts in Ada with statement counts in FORTRAN, we suspect they are roughly comparable measures of functionality (Reference 2). Assuming comparability, these results lead us to believe that Ada is somewhat more expensive than FORTRAN for conventional software development, but that reuse can lower the cost of an Ada delivery more than it can lower the cost of a FORTRAN delivery. We could conclude that FORTRAN is more cost effective for short-lived software but that Ada should be used for software that is likely to have a longer life through future reuse. We revisit this observation in the final section of the paper.

Shorter Cycle Time

Ada was also expected to lead to shorter cycle times or project durations. Figure 6 shows that this goal was met not only by the Ada projects but also by the FORTRAN projects. Again, the first high-reuse project in each language is the first project of each of the recent sets represented by the right-hand (darker) bars.

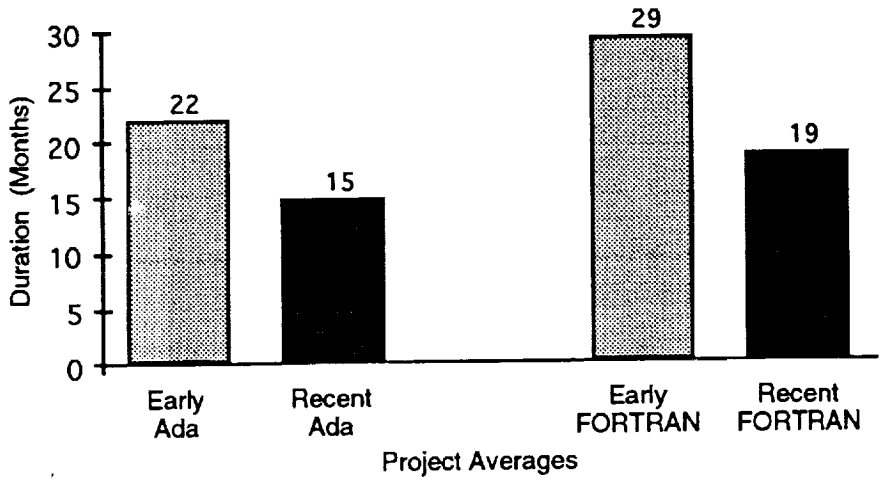


Figure 6. Average Project Duration: Before and After Reuse Process Adoption

The software development process did not change as suddenly as the reuse results, however. In fact, the schedule for the first high-reuse project in each language was more similar to earlier projects than it was to the subsequent high-reuse projects.[1] This is because the overall development process changed only after the EUVE project demonstrated that substantial savings could be achieved through large-scale reuse. To minimize risk, the first high-reuse project in each language was conducted using a more traditional schedule and staffing level. When management was able to observe the potential savings from reuse, procedural and scheduling changes were made to allow an expedited development process whenever high reuse was possible. So, whereas reuse can permit shortened project schedules, it is also necessary to accommodate this different behavior with an appropriately pared-down process. For example, the FDD now specifies a single design review in place of the traditional preliminary and critical design reviews whenever the majority of a new system can be constructed from existing code.

Reliability

The last explicit goal for the planned Ada transition was to increase the quality of the delivered systems. The density of errors discovered during development, measured on all FDD projects, is used to reflect system quality and reliability since quantitative operational data were not

---

[1] Because they behaved more like traditional projects, one might argue that the first high-reuse project in each language actually belongs to the early project set (represented by the left-hand bars of each pair). This would further accentuate the difference between early and recent projects.

collected.[2] Development-time errors are a useful reflection of quality because they reveal the potential for latent undetected errors and indicate spoilage and rework during development that, in turn, impact productivity and schedule.

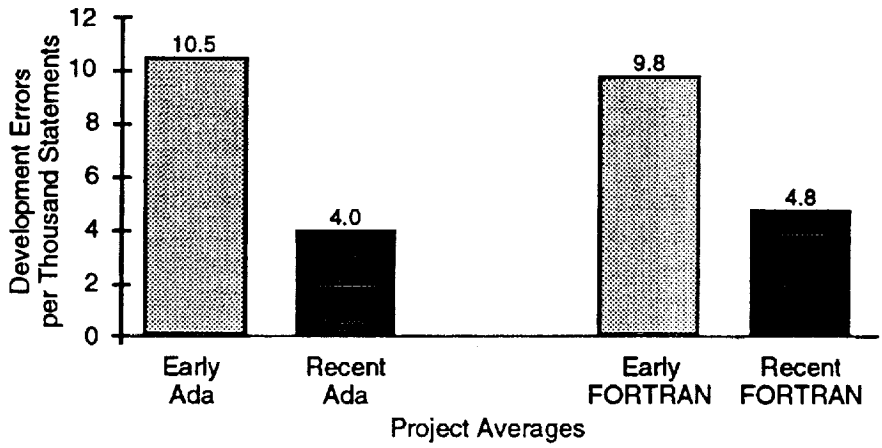


Figure 7. Reduction in Error Density in New or Modified Code: Ada and FORTRAN

The number of errors discovered per thousand statements of new and modified software before delivery is shown in Figure 7. Since the densities shown are based on only the new and modified code (verbatim reused code was not included in the denominator), we do not attribute these reductions to reuse. Instead, we attribute the reduced error rate to improvements in the development process that were instituted on all FDD projects during this period. These improvements included the use of object-oriented or encapsulated designs and the use of structured code reading and inspections. The fact that these process improvements were applied to projects in both languages is reflected by the similarity in the error-density reductions observed.

Process

An evolving development process was cited in the preceding discussion as being the reason for improvements in schedule and quality. We characterized the process by examining the distribution of effort across the various software development activities performed. The activity distributions shown in Figure 8 provide evidence that the more recent projects were conducted using a different process than the early projects. The figure shows the average number of staff-hours per project consumed by each of the four defined activities for software projects at the FDD. The dark bars for each activity show the averages for the first five Ada simulator projects, and the light bars show the average effort per activity for the five subsequent Ada simulators that achieved higher levels of reuse. (To fairly average the efforts for each activity among projects of varying sizes, the activity data for each project were first normalized by project size.) The savings exhibited for the later set of projects is due not simply to reuse alone

---

[2] So far, operational reliability, in terms of mean time to failure, has been adequate and it has therefore never become a measurement or improvement goal. The amount of maintenance effort expended on the operational systems is now being tracked, however.

but to the process change adopted to accommodate that reuse. The evidence for this is that the EUVE simulator (the sixth Ada project, see Figure 3) was the first to achieve a high level of reuse but, because it used a traditional process, its individual profile more closely resembled the earlier low-reuse group than the later high-reuse group.

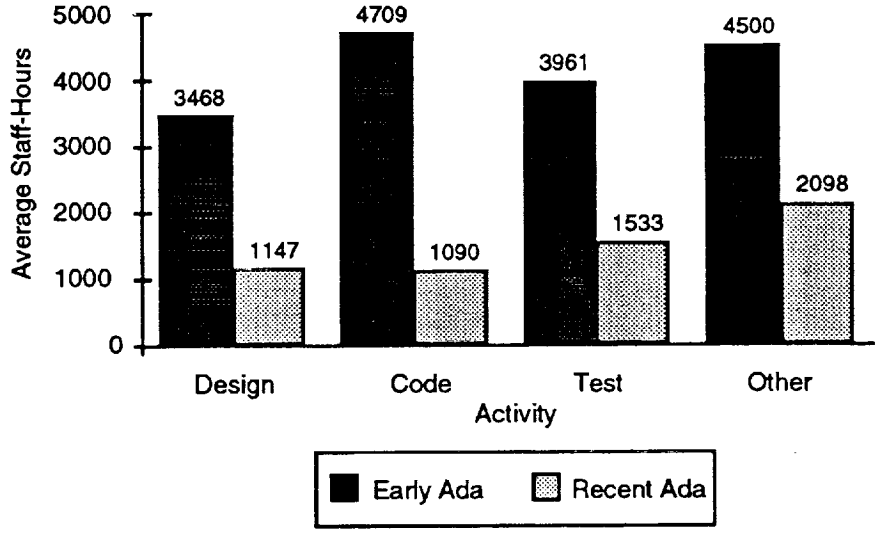


Figure 8. Average Effort by Activity: Early vs. Recent Ada Projects

Figure 9 shows the average effort by activity for the FORTRAN projects that were completed during the same period. Again, the effort magnitudes are normalized, and the projects are divided into an earlier group of lower reuse projects and a more recent group of higher reuse projects. As with Ada, a reduction in effort is shown for each activity when comparing low reuse with high reuse, although the net reduction is less in FORTRAN. Unlike the Ada results, however, most of the reduction occurs in the coding activity instead of being spread more evenly across every activity.

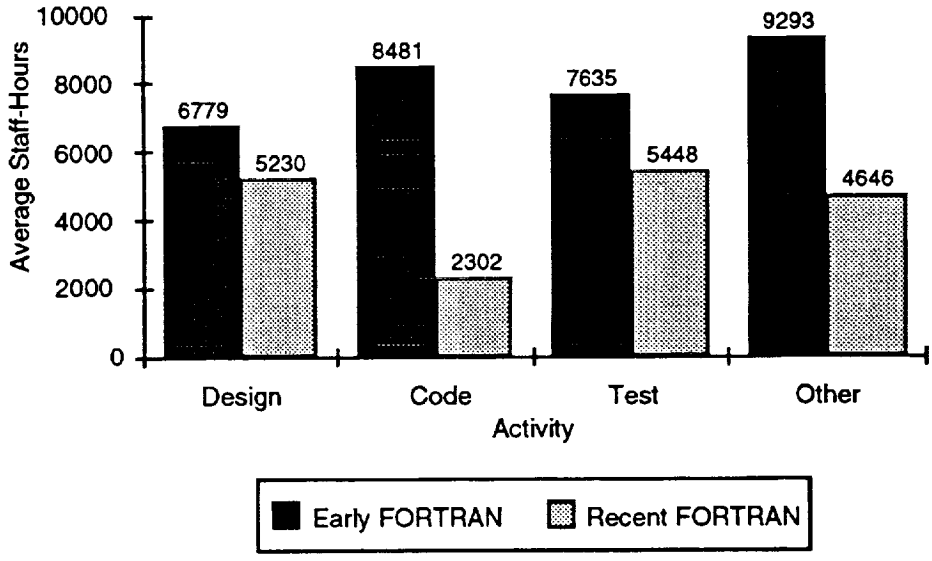


Figure 9. Average Effort by Activity: Early vs. Recent FORTRAN Projects

Notice that the shape of the activity distribution of the early projects in the FORTRAN set is virtually identical to the activity distribution of the early projects in the Ada set (the dark bars in Figures 8 and 9). However, as noted, the distributions for the recent, high-reuse projects differ between the languages. This suggests that the Ada and FORTRAN processes have each evolved in a different way even though they both had a common point of departure. The main lessons from this illustration are that the software process matured and improved during the time that Ada was used and that this evolution affected both the Ada work and the FORTRAN work, although in different ways. This process change is responsible for the reduced schedules shown in Figure 6. Also, the differing net reductions in effort reveal why Figure 5 showed a greater drop in cost for the Ada projects than for the FORTRAN projects when the benefits of reuse started to accrue.

Summary of the Comparisons

Quantitative data over the past 9 years show clear improvements attributable to the use of the Ada language in all of the initially specified goal areas. Many of these improvements were directly related to the increase in reuse. Interestingly, the FORTRAN systems showed comparable results over the same period, which leads to two possible interpretations. We could claim that because Ada did not provide a substantial improvement over FORTRAN, the FDD's continued involvement in the new language is unnecessary. However, Ada quickly did at least as well as the established language in the domain, and because it is a more modern language than FORTRAN, we could claim that Ada should be adopted for all future FDD development. Regardless of the interpretation chosen, it is safe to say that we found no quantitative evidence to indicate that Ada could not be used successfully on all FDD projects. The next section presents our additional findings beyond these quantitative results that affected the overall acceptance of Ada at the FDD.

Unforeseen Factors Impeding the Adoption of Ada

Given the encouraging Ada project results and the motivations of the forward-thinking managers who originally set out to transition to Ada, why hasn't the FDD successfully adopted Ada? As stated earlier, only a fraction of all new software developed at the FDD is in the Ada language. Our investigation discovered several unforeseen factors that have impeded the FDD's transition to Ada. These factors were poor initial system performance, the lack of adequate tool support, and developer bias. This section discusses these factors and their impact on the transition to Ada.

Performance

System performance was not an explicitly stated goal for the programs developed in Ada, but it turned out to be a major issue. By 1985, the programmers in the FDD had achieved such proficiency with FORTRAN software design and implementation that even the most complex flight dynamics systems performed adequately without paying any special attention to performance. Thus, performance had become an implicit consideration and was not addressed in software requirements, designs, or test plans.

Figure 10 depicts the relative response times of the delivered simulators between 1984 and 1993. A smaller response time indicates better performance. The figure reveals that the first

Ada simulator performed very poorly compared with predecessor FORTRAN simulators. Because Ada language benchmarks had shown that Ada executed as fast as equivalent FORTRAN programs and because performance was not an explicit goal, developers of the first Ada project paid little attention to performance. Instead they focused on learning the language and developing reusable software. It should come as no surprise that novice users of this fairly complex language did not produce an optimum design or implementation. But, because this system was delivered for operational use, the FDD users' first encounter with an Ada system was negative. This feeling was compounded by the fact that, because of scaled-down processing requirements, the FORTRAN simulator delivered immediately before the first Ada simulator was the fastest simulator ever delivered.

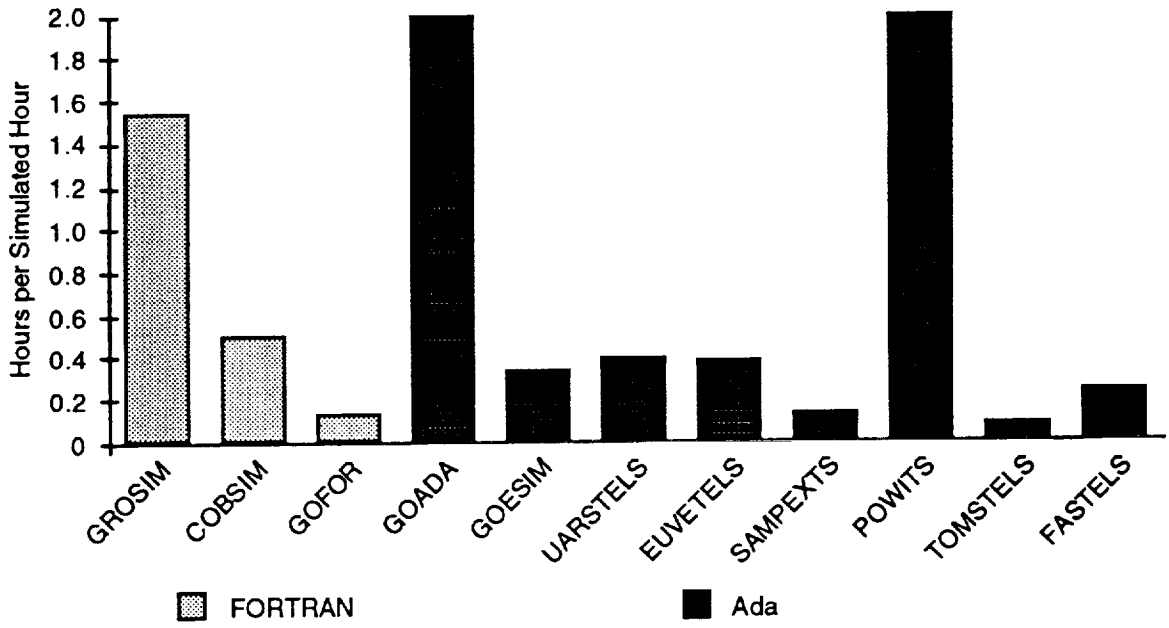


Figure 10. Relative Response Times for Ada and FORTRAN Simulators

In 1990, the FDD conducted a performance study (Reference 6) to determine why the Ada systems executed so much more slowly than the fastest FORTRAN system. The study discovered that some of the coding techniques practiced in FORTRAN to achieve high efficiency actually worked against efficiency in Ada and that some of the data structures around which the designs were built were handled very inefficiently by the DEC Ada compiler. The study resulted in a set of Ada efficiency guidelines (Reference 7) for both design and code that is now being followed for all new Ada systems. Interestingly, in order to follow those guidelines, the last two Ada projects in Figure 10 had to forgo a certain amount of reuse. (The slower POWITS simulator was completed before these guidelines were available and also had considerably more complex processing requirements.) As shown in the figure, the typical Ada simulator now performs better than most of the earlier FORTRAN simulators. However, first impressions are very important, and the perception of many of the FDD programmers and users is that Ada still has performance problems and that systems demanding high performance should not be implemented in Ada.

Ada Development Environments

Finding adequate vendor tools to support Ada development in the FDD was a major obstacle. In 1985, when the FDD began its work with Ada, most computer vendors were actively developing Ada compilers and development environments. The FDD believed that vendor tools would be widely available within a few years. But, consistently usable Ada development environments and reliable Ada compilers did not become available across the platforms used to develop and execute FDD software systems.

All the Ada projects included in this study were developed using DEC Ada on VAX minicomputers that were rated by FDD developers as having a good set of Ada development tools to facilitate the development process. However, 80 percent of the software developed in the FDD must execute on the standard operational environment, which is an IBM mainframe. And traditionally, FDD systems have been developed on their target platforms. Unfortunately, an adequate Ada development environment for the IBM mainframe was never found.

The FDD conducted several compiler evaluations and a portability study between 1989 and 1992, all of which declared the IBM mainframe environments unfit for FDD software development. In 1989, the FDD evaluated three compilers (Reference 8) and selected one for purchase and further study. Somewhat discouraged by this study which rated the best compiler as having only marginal performance for flight dynamics computations and no development tools, the FDD investigated an alternative approach. Since Ada was touted to be highly portable, they conducted a portability study to determine whether FDD systems could be developed in Ada on the VAX and then transported to the mainframe for operational use. This study (Reference 9) ported one of the existing operational Ada simulators from the VAX to the IBM mainframe using the Alsys IBM Ada compiler, version 3.6. The study found that relatively few software changes were required and that the resulting system performed adequately on the mainframe, but that rehosting was extremely difficult because of compiler problems and the lack of diagnostic tools and library management tools. Although rehosting the system required only a small amount of effort, it took nearly as much calendar time as was needed to develop the system from scratch.

In the fall of 1992, the FDD again conducted a compiler evaluation on what were supposed to be greatly improved products. This study (References 10 and 11) selected a different compiler than the earlier study, using the ported simulator as one of its benchmarks. Although the chosen compiler performed better than other candidates and was accompanied by a limited tool set, the study warned against using it to develop real-time or large-scale FDD systems because of its inefficient compiling and binding performance, immature error handling, and poor performance of file input/output. Finally, late in 1993, the FDD achieved some limited success developing a small utility in Ada (the FAST General Torquer Command Utility) on an IBM RS-6000 workstation and porting the software to the mainframe.

Figure 11 (a simplification of Figure 2) shows a sharp decline in the amount of Ada development late in 1990. It was at this point that the FDD had planned to begin developing parts of the larger ground support systems in Ada on the mainframes. But results of the early Ada compiler evaluation and the portability study made it clear that developing on, or even developing elsewhere and porting to, the mainframes was not feasible. Thus, continued growth in Ada stalled.

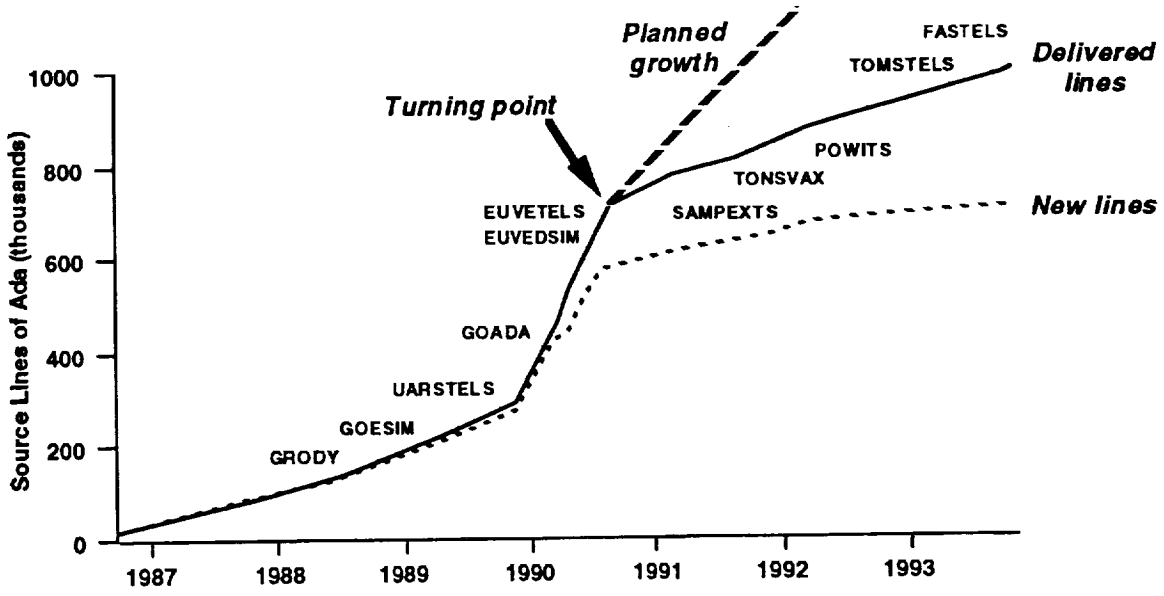


Figure 11. Drop in Ada Growth Coincides With Trouble Migrating to Mainframes

At the same time, the FDD's simulation requirements changed, reducing the number of simulators required to support each spacecraft mission from two to just one.[3] This change resulted in a reduction in the amount of software targeted to be developed in Ada. The net result was a significant reduction, instead of an increase, in the rate of Ada software delivery. The drop in Ada development is even more dramatic when you eliminate the amount of reused software and consider only the investment in new Ada code. The flatter dashed line below the curve for cumulative delivered size in Figure 11 shows only the number of new and modified lines being developed.

The unavailability of an adequate Ada development environment on the IBM mainframe was clearly a significant stumbling block to the FDD in its transition to Ada. We feel that as long as the mainframes remain the principal operational environment at the FDD and as long as mainframe development or deployment of Ada systems remains awkward, there appears to be no straightforward way to increase the use of Ada within the Division.

In addition to its disappointment with the mainframe development environments, the FDD also experienced only limited success in using Ada to develop an embedded system. As part of a separate research and development effort, the FDD developed an embedded application on a Texas Instruments 1750A machine using the Tartan Ada compiler. Unfortunately, the two-vendor situation led to interface problems between the hardware and software; the lack of diagnostic tools contributed to insoluble problems that resulted in an end product with reduced capability. This experience contributed to the general feeling among the FDD developers that there was not yet a satisfactory level of vendor support for Ada development.

---

[3] Before this point, both a telemetry simulator and a dynamics simulator were developed for each mission. Since 1990, only an enhanced telemetry simulator has been required.

Developer Perspective

As with any technology infusion effort, developers' bias for or against a technology can significantly facilitate or impede the transition. To get a reading on the current perspective of the developer community, we identified and interviewed 35 FDD developers who have been trained in or have developed systems in Ada. Each developer was asked which language they would choose for the next simulator project and which language they would choose for the next ground support system and why. Figure 12 shows their responses.

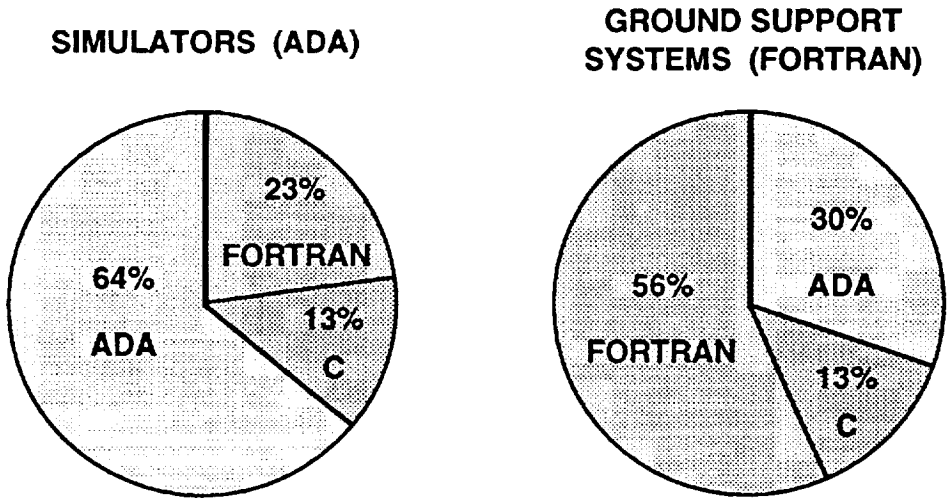


Figure 12. Developer Preferences for Programming Language

Most agreed that Ada should be used for the next simulator, whereas FORTRAN should be used for the next ground support system, citing the availability of reusable components and architectures as the deciding factors. But, also notice that 25 to 30 percent of the developers seemed to have a language bias because they chose the language not customarily used for each type of application. The 23 percent who preferred to use FORTRAN for simulators cited the complexity of the Ada language and poor performance as reasons to abandon Ada, while the 30 percent who preferred to use Ada to build the next ground support system felt that Ada was a better language for building larger systems.

Nearly all the developers pointed out that adequate tools are essential for efficient and accurate development using Ada, whereas FORTRAN development can be accomplished with little or no tool support. However, the two most enthusiastic Ada developers felt that they were sufficiently proficient with the language to overcome the lack of tools on the mainframe computers. Interestingly, at the other extreme, several the developers did not care one way or the other about which language they used for software development; two developers specifically commented that "Ada is just another language."

The two significant minority groups who were the most opinionated about language use, one in favor of Ada and the other opposed, have been fairly vocal and forthcoming with their views over the past several years. Thus, there may be an observable effect from these biases on the remaining group of developers who have not yet been trained in or exposed to Ada. We plan to investigate these possible effects through additional developer surveys before this study is concluded.

CONCLUSIONS AND RECOMMENDATIONS

In 1985 Ada was arguably more than just another programming language. However, by exposing the organization to the concepts of information hiding, modularity, and packaging for reuse, that which was "more than a language" was adopted, to the extent possible, by the FORTRAN developers as well as by the Ada developers. We hypothesize, and in fact have anecdotal evidence to support the theory, that Ada served to catalyze several language-independent advances in the ways in which software is structured and developed across the organization, and that these benefits have been institutionalized by process improvements.[4] Because many of the intended benefits of Ada have already accrued at the FDD and because the mainframe obstacle continues to hamper the complete adoption of Ada, we recommend that the FDD not mandate the use of Ada for all software development at the FDD.

However, we also recommend that the FDD continue to use Ada wherever there is a clear or anticipated advantage. This would include not only the continued use of Ada on satellite simulators but also the use of Ada on portions of any other projects that are expected to be long-lived and can be developed and deployed on an Ada-capable platform. As the FDD migrates away from mainframes and toward workstations, this will be an increasingly large segment of the software developed. Over the long term, Ada is likely to be a good candidate for future versions of the large reusable software libraries that are currently written in FORTRAN and maintained by a separate group of experts who are constantly augmenting the code's function in order to keep up with the needs of the client projects. Ada can be used to implement those subsystems, along with many other basic domain functions, as sets of separable and more maintainable abstractions, which would eliminate the high coupling found in the FORTRAN versions.

Finally, it is valuable to examine some of the original objectives behind the development of the Ada language. One of the major motivations for the DoD to develop and adopt Ada was to provide a common language across many projects, thus enabling the portability of programs, tools, and personnel. However, such commonality is not as important at the FDD where most of the software is already written in a single high-order language and is both developed and operated on a single platform by a seasoned team with low turnover. Besides providing the DoD with a common language, Ada was specifically designed to be beneficial for large system development and to be cost effective for systems that must be maintained over long periods of time. Neither of these situations describe the context in which Ada has been used at the FDD. If the FDD Ada projects had been large (closer to a million lines of code, for example) then we suspect that the use of Ada would have been a more important factor. Also, if the systems were long-lived, with maintenance cycles that were substantially longer than the development cycles, or even if the longer-lived ground support systems had been able to use Ada, we suspect that we would have observed more of the advantages of Ada that would not have been mimicked by the FORTRAN projects.

---

[4] These opinions were commonly held by the project managers of several of the Ada and FORTRAN projects included in this study.

THE AUTHORS

John Bailey is an independent consultant in the areas of software measurement and the Ada language. Sharon Waligora is a member of the Software Engineering Laboratory at Computer Sciences Corporation and has experience managing and working on the Flight Dynamics Division's software projects. Mike Stark is an employee of the Flight Dynamics Division at Goddard and also has experience leading and participating in many of the FDD Ada and FORTRAN projects.

REFERENCES

1. NASA/GSFC Software Engineering Laboratory, SEL-82-1206, Annotated Bibliography of Software Engineering Laboratory Literature, L. Morusiewicz and J. Valett, November 1993.

2. Institute for Defense Analysis (IDA), IDA Paper P-2899, "*Comparing Ada and FORTRAN Lines of Code: Some Experimental Results*," T. Frazier, J. Bailey, M. Young, November 1993.

3. NASA/GSFC Software Engineering Laboratory, SEL-81-305, *Recommended Approach to Software Development*, L. Landis, S. Waligora, F. McGarry, et al., June 1992.

4. ___, SEL-81-305SP1, *Ada Developers' Supplement to the Recommended Approach*, L. Landis, R. Kester, June 1992.

5. ___, SEL-91-006, *Proceedings of the Sixteenth Annual Software Engineering Workshop*, "Experiments in Software Engineering Technology," F. McGarry and S. Waligora, December 1991.

6. ___, SEL-91-003, *Ada Performance Study Report*, E. Booth and M. Stark, July 1991.

7. Goddard Space Flight Center (GSFC), Flight Dynamics Division, 552-FDD-91/068R0UD0, *Ada Efficiency Guide*, E. Booth (CSC), prepared by Computer Sciences Corporation, August 1992.

8. ___, "*Ada Compilers on the IBM Mainframe (NAS8040) Evaluation Report*, L. Jun, January 1989.

9. NASA/GSFC Software Engineering Laboratory, SEL-90-003, *A Study of the Portability of an Ada System in the Software Engineering Laboratory*, L. Jun and S. Valett, June 1990.

10. Goddard Space Flight Center (GSFC), Flight Dynamics Division, "*IBM Ada/370 (Release 2.0) Compiler Evaluation Report*," L. Jun, September 1992.

11. ___, "*Intermetrics MVS/Ada Version 8.0 Compiler Evaluation Report*," L. Jun, October 1992.

Impact of Ada in the Flight Dynamics Environment: Excitement and Frustration

December 2, 1993

John Bailey, *Software Metrics, Inc.*

Mike Stark, *NASA/Goddard*
Sharon Waligora, *Computer Sciences Corporation*

SMI

Independent Assessment

- Flight Dynamics Division (FDD) began investigating Ada in 1985
- Expected to transition completely to Ada within 10 years
- Why is only 15% of new code being written in Ada today?
- What is the future of Ada in the FDD?
  - Mandate?
  - Abandon?
  - Change expectations?
  - Study further?

SMI

Why Use Ada in FDD?

"Ada is more than just another language"

- Would lead to a major cultural change
- Would drive an integrated well-defined software engineering process
- Would help us build better products
  - Reduce life-cycle cost
  - Shorten project duration
  - Reduce number of errors
  - Increase manageability of software

SMI

FDD Environment

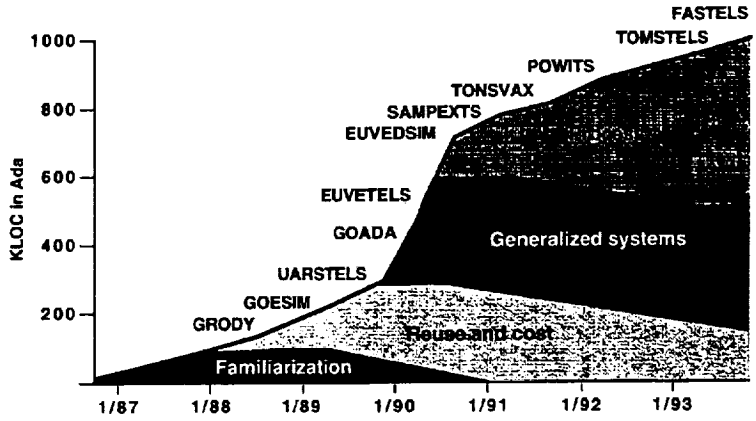
Organization Staff Level > 250

Characteristics	Applications	
	Mission Support	Simulators
Computing Environment	IBM mainframes	VAX
Language	FORTRAN	Ada
Typical System Size	200 KLOC	60 KLOC
Percent of Software	80%	15%

SMI

Focus of Ada Study Evolved

FDD Ada Experience



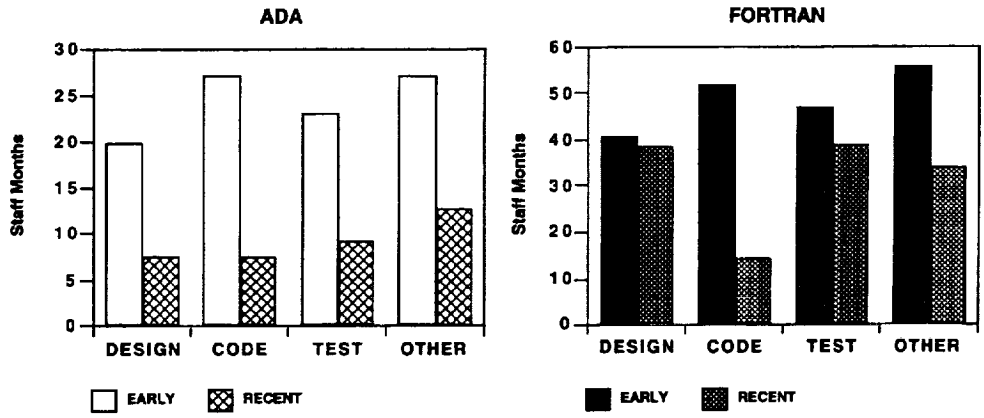
SMI

The Excitement: Data Show Promise

- **Process Improvement**
- **Product Improvement**
  - **Reuse**
  - **Cost**
  - **Project duration**
  - **Reliability**

SMI

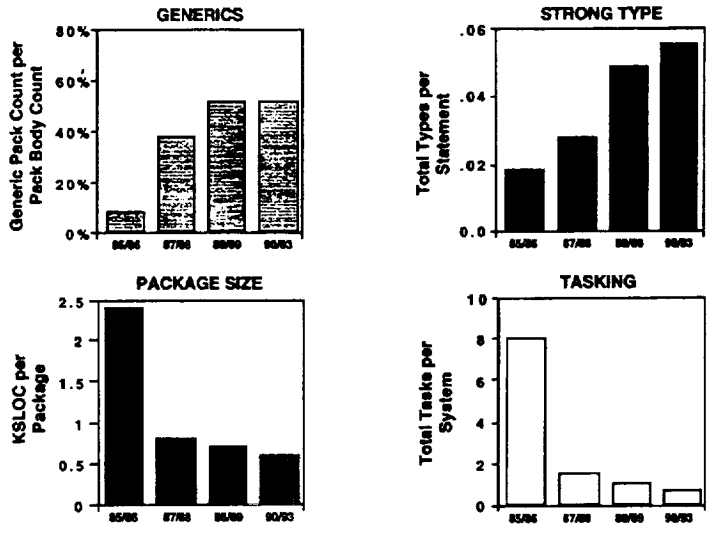
Evidence of Process Change Over Time



Activity distribution shows process differences

SMI

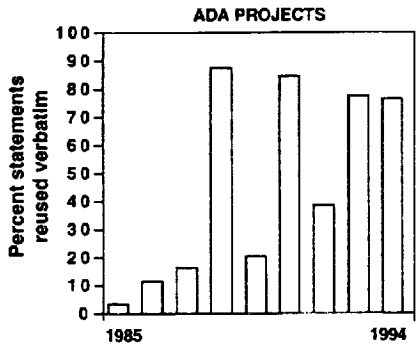
Maturing Use of Ada



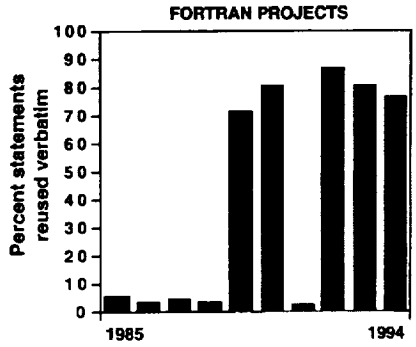
Use of new features stabilized after experience with domain

SMI

High Verbatim Reuse Achieved in Both Languages



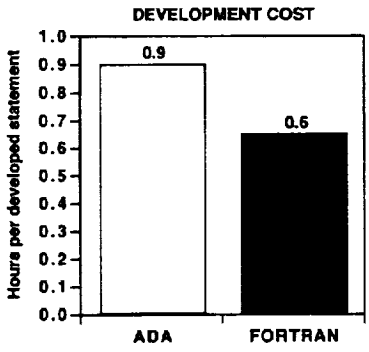
Ada approach uses generics parameterized for mission-specific functionality



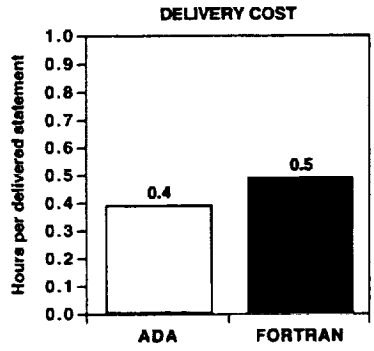
FORTRAN approach uses separately maintained utilities, with mission-specific functionality added as needed

SMI

Cost To Develop and Deliver



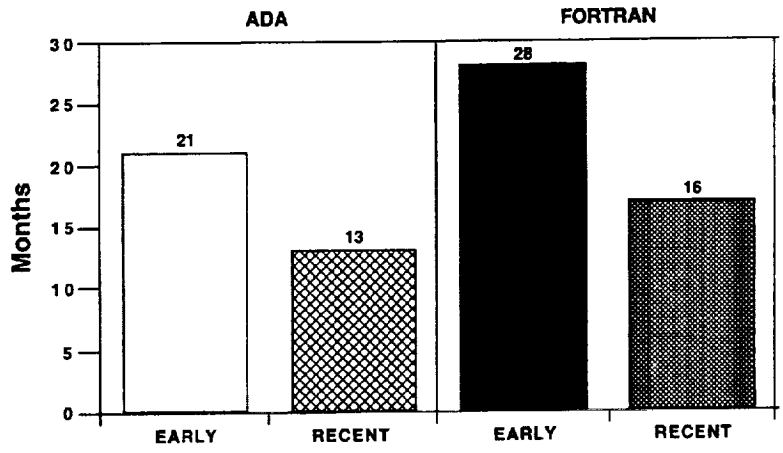
Cost to develop a statement of Ada is higher



Cost to deliver a statement of Ada is lower

SMI

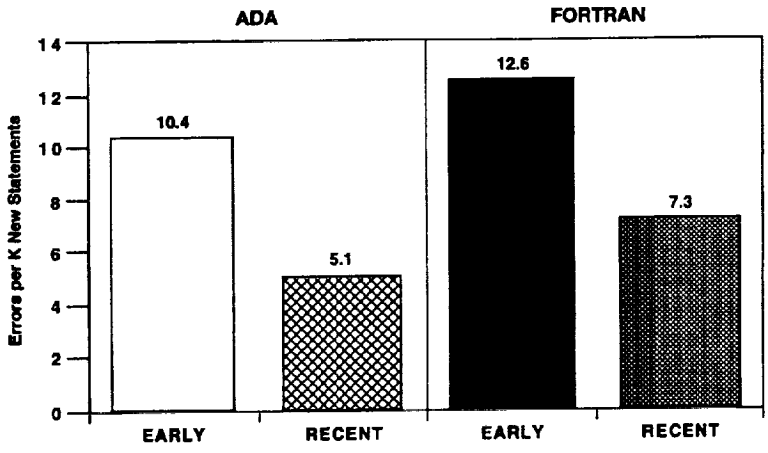
Reuse Shortens Project Duration



Both Ada and FORTRAN show 38% reduction due to reuse

SMI

Reliability of Systems Increased



Lower error rates are observed for both languages

SMI

Measurements Show Process and Product Improvements

- Ada systems showed improvements in initial goal areas
- Many achievements are tied to reuse
- Lower error rates in new code are attributed to process changes
- FORTRAN systems showed comparable results over same time period

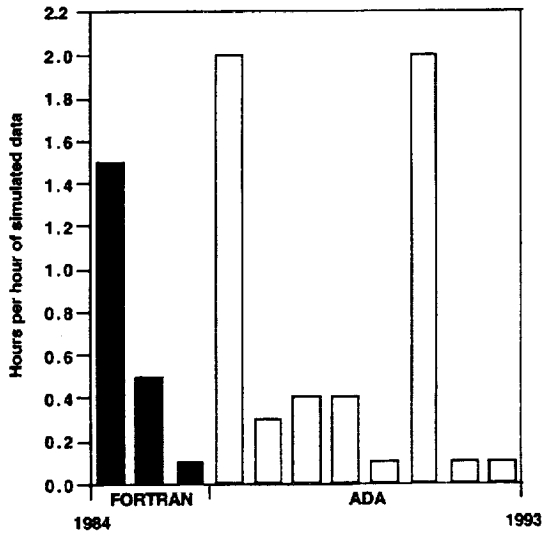
SMI

The Frustration: Other Factors Derail Progress

- Performance of early systems
- Ada development environments
- Technology transition

SMI

Performance: First Impressions Count



Weak performance of early simulators gave Ada a bad reputation

SMI

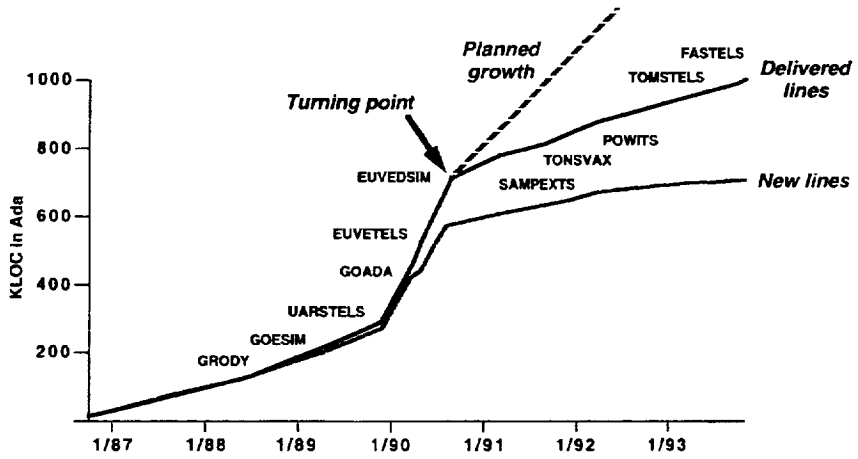
Ada Development Environments

- VAX Ada
  - Adequate performance
  - Good tools
- IBM mainframe
  - Very poor usability
  - Limited set of immature tools
- Tartan/1750A
  - Separate vendors for hardware and software
  - Major hardware/software interface problems
  - Required assembler-level debugging

Limited vendor support for Ada environments hampered efforts

SMI

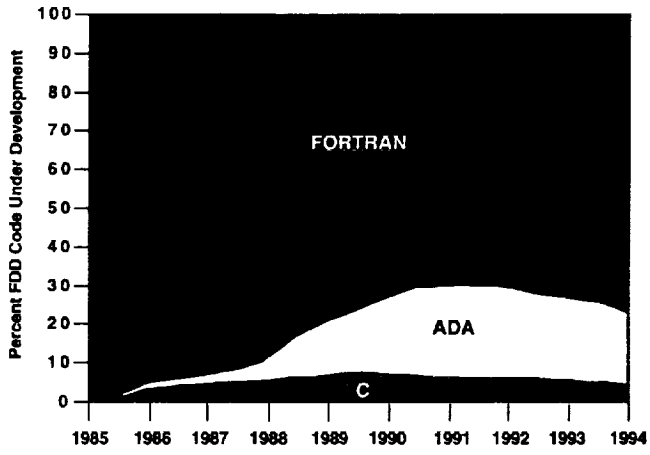
Mainframe Environment Problems Deterred Growth



Even today, Ada cannot be used on the FDD (mainframe) operational environment

SMI

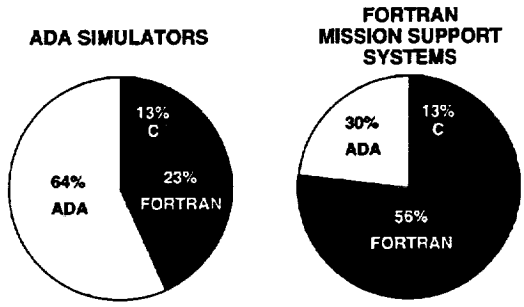
Technology Transition to Ada Has Slowed



Use of Ada drops as a result of deterrents

SMI

Developers' Preferences



- Interviewed 35 developers with Ada exposure
- Reuse was main driver for language preference
- Ada has advantages but requires more tool support
- "Ada is just another language"

SMI

Process or Language?

- Data show few quantitative differences between Ada and FORTRAN
- Process is more significant factor than language
- Ada concepts (generalization, OOD, domain analysis, information hiding) lay foundation for broad process improvements
- Structured environment and strong process management institutionalize improvement

SMI

Future of Ada at FDD

omit

- Ada should not be mandated at the FDD
  - No pressing need for common language
- Ada should be used as any other method or tool

SMi

10015848-press 21