00

N91-10613

3/8

Ada Software Productivity in Prototypes: A Case Study

JJ574450

Jairus M. Hihn
Hamid Habib-agahi
Shan Malhotra

Jet Propulsion Laboratory
California Institute of Technology

ABSTRACT

This paper is a case study of the impact of Ada on a Command and Control project completed at the Jet Propulsion Laboratory (JPL). The data for this study was collected as part of a general survey of software costs and productivity at JPL and other NASA sites.

The task analyzed is a successful example of the use of rapid prototyping as applied to command and control for the US Air Force and provides the US Air Force Military Airlift Command with the ability to track aircraft, air crews and payloads worldwide. The task consists of a replicated database at several globally distributed sites. The local databases at each site can be updated within seconds after changes are entered at any one site. The system must be able to handle up to 400,000 activities per day. There are currently seven sites, each with a local area network of computers and a variety of user displays; the local area networks are tied together into a single wide area network.

Using data obtained for eight modules, totaling approximately 500,000 source lines of code, we analyze the differences in productivities between subtasks. Factors considered are percentage of Ada used in coding, years of programmer experience, and the use of Ada tools and modern programming practices.

The principle findings are the following. Productivity is very sensitive to programmer experience. The use of Ada software tools and the use of modern programming practices are important; without such use Ada is just a large complex language which can cause productivity to decrease. The impact of Ada on development effort phases is consistent with earlier reports at the project level but not at the module level.

Introduction

The Economics Group at JPL has been involved in the collection and analysis of software cost and productivity data for the past three years. The NASA Historical Database contains data for over 100 subsystems including 10 different projects. [Economics Group 1989] The JPL Software Database currently contains data for 4 projects with 39 subsystems.[SORCE/Economics Group 1988] During the coming year data on seven more projects will be collected. A relatively unique feature of these databases is that they contain data on all the subsystems of each project for which information could be obtained. Most software databases used for research contain only one or two observations from any one project. The advantage is that we are able to control for differences between projects which are not directly measured by the specific database fields and also can also analyze within project variations in effort and productivity. The disadvantage is that a larger number of observations must be collected to get a sufficient number of independent data points for statistical analysis.

The data collected is primarily based on the COCOMO definition of a software environment. [Boehm, B. 1981] Table 1 lists the cost driver contained in the database which describe the environment. The database also includes size, measured by executable source lines of code adjusted for inherited and modified code, and effort, measured by work months. The portion of the life cycle for which effort figures have been collected includes from the requirements analysis phase through test and integration. Sustaining engineering and the systems engineering effort to develop the requirements are nor included. However systems engineering effort spent on requirements design updates and formal design reviews is included. Two estimates of effort were collected. Technical effort figures gathered from interviews with the technical leads, estimates direct effort by programmers and the technical managers. Implementation effort figures derived from the task management office, include all labor charges to the project from the task manager down. The non-direct labor charges are distributed across the subsystems on a proportional basis. These charges include integration and validation testing, documentation and management labor time. Implementation effort also includes secretarial time which could not be separated out. Effort figures do not include upper level project management or system engineering previous to the SRR.

Table 1
Database Description

Product Attributes
Required reliability
Software complexity
Database size

Personnel Attributes
Analyst ability
Analyst experience
Programmer ability
Language Experience
Virtual Experience

Computer Attributes
Time constraints
Storage constraints
Host volatility
Turnaround time

Project Attributes
Software tools
Modern programming practices
Schedule

The average productivities in the NASA Historical Database are 1.5 to 3.5 SLOC per day for flight software and 7 to 10 SLOC per day for ground based software. There were a few subsystems which reached approximately 14 SLOC. In the JPL Software Database, the average productivity ranged from 6 to 18 for 3 DOD projects and one ground data capture project. There are two command and control projects which had the highest productivities of the projects we have studied. Project 1 used Ada and rapid prototyping to reach a implementation productivity of 17.9 SLOC/ work day. Project 2 which was very similar to Project 1 did not use Ada and had an implementation productivity of 13.5 SLOC/ work day. The purpose of this study is to attempt to isolate the impact of Ada versus the impact of software tools, modern programming practices and other environmental factors on productivity.

Project Description

The US Air Force Military Airlift Command (MAC) runs one of the largest airlines in the world. Scheduling problems are accentuated because flights, crews, and payloads can be changed at any time in order to meet political and military objectives. MAC is in the process of automating its command and control system by replacing its current scheduling system, based on grease boards and the telephone, with a network of workstations supporting a replicated database with real-time displays. Two major components of MAC's Command and Control Automation Project are being completed by JPL. Project 1 supports the vertical command and control operations, and Project 2 supports the actual execution of tasks. Project 1, a successful example of the use of rapid prototyping, consists of a globally distributed replicated database with sites from Germany to Hawaii.

Developed as a prototype which became an operational system, Project 1 had an unusual software life cycle for a delivered system. JPL was required to develop Project 1 within two years at minimal cost. The functional requirements were vague becauso the sponsor was not very computer literate. The project manager compensated for these factors by waiving many of the standard formal design, documentation, and testing requirements and by developing a very close working relationship with the sponsor. The final requirements evolved as part of a joint effort between the project team and the sponsor. Detailed documentation, except for the user's guides, could be written after the project team and the sponsor had agreed that the system was working.

Project 1 consists of five application subsystems and three support system subsystems. The applications support the following five MAC functional groups: Current Operations (DOO), Transportation (TR), Command and Control (DOC), Logistics (LRC), and the Crisis Action Team (CAT). The software work breakdown structure is similar to the functional breakdown; therefore, the descriptions which follow of functional groups also serve as descriptions of corresponding software tasks. DOO performs flight scheduling and resource planning. TR is responsible for personnel ticketing and cargo loading and unloading. DOC monitors the progress of each flight. When en route mechanical failures occur, LRC provides information which assists in the prompt servicing of debilitated aircraft. CAT controls system responses in the event of a threat or emergency.

System support for Project 1 resides in three subsystems: Graphics, Operating System Shell, and Database. Graphics produces a graphical display of database information while allowing the user to manipulate screens via a user interface. Operating System Shell provides an interface to VMS OS, network commands, and low level VMS functions. Database supports database design and control.

Table 2

Development Data

Subtask	Size (KSLOC)	Technical Effort (Work Months)	Implementation Effort (Work Months)	Technical Productivity (SLOC/day)	Implementation Productivity (SLOC/day)
Application Software					
DOC	72	118	207	32	18
DOO	115	140	245	43	25
LRC	45	28	49	84	48
CAT	23	36	63	34	20
TR	70	60	105	61	35
System Software					
Graphics	20	72	145	15	8.3
Common	110	258	453	22	13
Database	37	110	193	17.7	11
Total	492	822	1,460	31.5	17.9

The development data collected was based upon the status of the project in January 1988 which was before the software system was actually converted into a formal product. The total size was approximately 500,000 source lines of code.[1] The sizes of the modules range from 20,000 to 115,000 source lines of code. The code count is based on executable source lines of code; the size figures do not include comments or blank lines.

The productivity figures for the Project 1 subsystems are presented in Table 2. The average technical productivity of Project 1 as a whole was 31.5 source lines of code per day; the average total productivity was 17.9 source lines of code per day. At the time of final delivery of the system implementation productivity had increased to an average of 20 SLOC/ work day. This occurred even though documentation and testing effort increased significantly during the last release. This is most likely a result of the staff being further up on the learning curve with respect to Ada and the application domain. Among

[1] At final delivery Project 1 will have reached approximately 750,000 source lines of code.

the systems tasks, total productivities averaged 11 and ranged from 8 to 13 source lines of code per day. The application tasks had total productivities averaging 25 and ranging from 18 to 48 source lines of code per day. In general, application software is associated with higher productivities than system software because application software is less embedded and usually does not have to incorporate low level implementation details.

Table 3 summarizes the values of the environmental factors included in the database for Project 1. However, the table shows that experience and capability were rated high; requirements volatility was rated low; and the use of modern programming practices and software tools was extensive throughout the project.

Table 3
Project 1
Development Environment

Product Attributes	Low to Nominal
Computer Attributes	Low to Nominal
Personnel Attributes	High
Project Attributes	High

ANALYSIS

Project 1 developers achieved higher total productivity than the average NASA project teams developing ground software. Several factors combined to permit this achievement: the ability to match highly qualified personnel to the task needs, the use of a prototyping methodology, the organizational structure of the development team, an abundance of development tools, excellent communications with the sponsor, development team cohesiveness, and the use of Ada.

The development environment contributed to the high productivity of the project staff. The implementation managers were able to match skills and project needs with programmers whose capability and experience were well above average. Project 1 was developed as an incremental prototype; the development strategy cut the standard development life cycle. User's guides were written in parallel with the software. A single design document was written at the end of the project which was the equivalent of an FRD, FDD, SRD, and SDD combination to assist during the sustaining engineering phase. In the testing phase a formal independent validation and verification was omitted; and there were no formal preliminary and critical design reviews by an external organization. However, there was a formal internal review prior to each major software release. The small overall staff size facilitated open communication within groups, between groups, and with the sponsor. The sponsor provided ample hardware which was appropriate to each task. Finally, the majority of the tasks were of moderate difficulty or complexity.

One other factor that potentially contributed to the high productivity of Project 1 was the use of Ada. At the time of the initial survey fifty percent of the total code was Ada and varied from 0 to 90% across the subsystems. When the project started about half of the programmers had an average of 1 year experience with Ada and the rest had no experience. A few had the maximum possible experience of about 2 to 2.5 years. There was no formal requirement that Ada had to be used. In the early 1990's Ada will be a more mature language, but this level of staff quality was the best that could be hoped for when software development began two years ago.

Ada advocates claim that the proper use of Ada, with its software tools, strong type checking, and support of modern programming practices, increases programmer productivity by over 100% and decreases program maintenance costs[Royce, W. 1987].

It is difficult to test these claims, however, because one must be very careful when comparing the productivities of programmers coding in different languages. In particular, Ada has several characteristics which can cause an Ada program to have more or fewer lines of code than other third-generation language programs with the same functionality. Ada's syntax for using objects can inflate an Ada program's code count. On the other hand, Ada's ability to use generic procedures can deflate Ada's code count, since a generic procedure would have to be written a number of times in a third generation language. A recent survey found that the effect of Ada on code count depends upon the application: business and scientific applications tend to result in larger Ada code counts whereas avionics and automation projects tend to have smaller Ada code counts.[Reifer, D. 1988]

Accurate measurement of the impact of Ada on productivity requires that major differences between organizational structures also be isolated. When subsystems of very different projects are compared environmental differences not captured in the data can arise. These differences especially relate to environmental factors such as communication between sponsor and contractor and cohesiveness of the programming teams. The result is very large variances in the data; conceptually the problem is that of comparing 'apples and oranges'.

The results of productivity comparisons between different projects and especially between languages is very sensitive to both the type of application and unexplained environmental factors. To reduce the impact of these problems we will emphasize comparisons between modules with similar amounts of Ada and comparisons between projects that are very similar in nature. The other project, Project 2, that will be referenced in the analysis is also a command and control task performed under the same project office at JPL and also for MAC. Both tasks were eventually housed in the same building and both were prototypes at the time of this survey.

Figure 1
Implementation Productivity Unadjusted

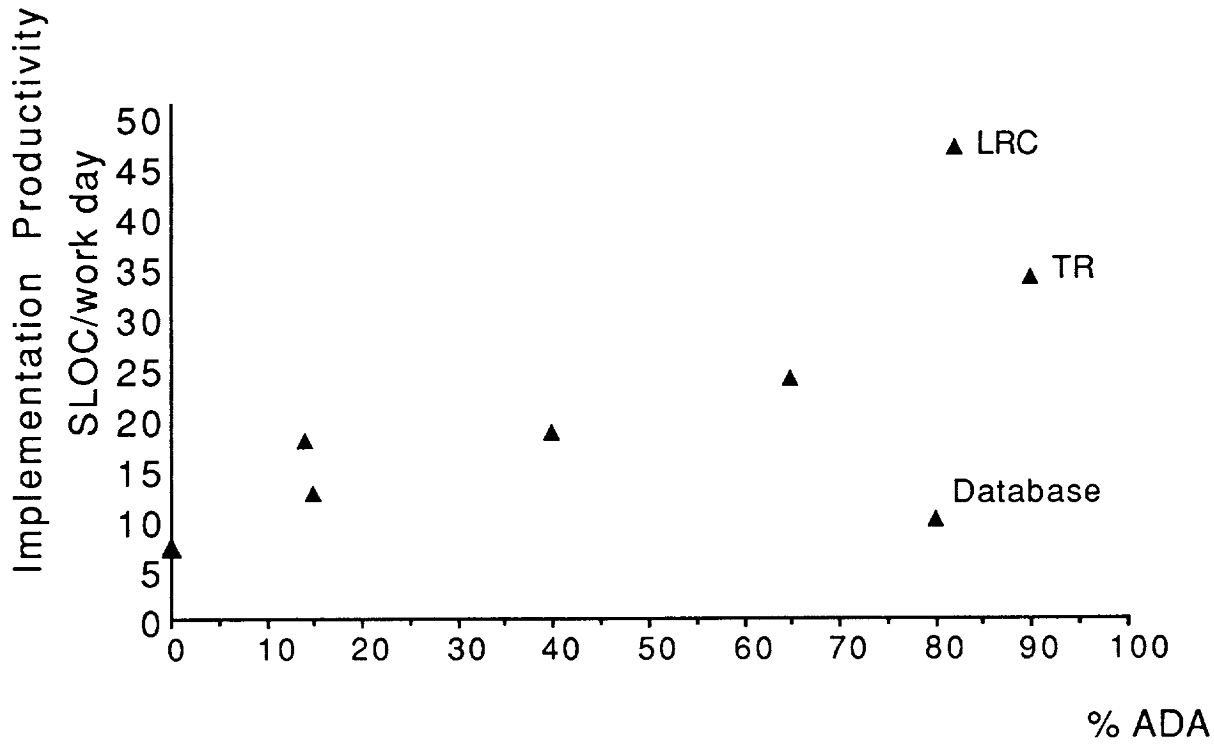


Figure 1 plots productivity, (SLOC/implementation effort)/19, against %Ada, the percent of code in Ada for a module.[2] The graph is suggestive of a positive correlation between the percent of a subsystem's code written in Ada and the productivity of that subsystem which would represent the combination of the impact of Ada and the cost of mixing languages. Comparing the average productivity of those modules with less than 50% Ada to those with greater than 50% Ada one may be tempted to draw the conclusion that use of Ada increases productivity by about 15 SLOC/day which would be close to a 100% increase. However there are many other differences between these subsystems which also impact productivity and these must be identified in order to isolate the actual impact of Ada on productivity.

For example, compare the productivities of subtasks with similar percents of code written in Ada. LRC, TR and Database are three such modules. Programmer experience and the use of modern programming practices and tools are significant differences between these subsystems. At the time of the survey the LRC technical lead, which achieved the highest productivity, had 2.5 years of experience coding in Ada and six years of experience object-oriented design. The nature of the LRC task allowed the team to use objects extensively. The LRC staff also consistently employed modern programming practices and software tools. The productivity of the TR team

[2] 19 represents the actual number of work days in a month when discounting for holidays, sick days and general meetings. [Boehm, B. 1981]

was lower than that achieved by the LRC staff; the TR staff did use modern programming practices and tools, but the TR programmers, with one to two years of experience coding in Ada, were less experienced than the LRC team members. The Database team were less productive than either the LRC or TR teams. Database had zero years Ada experience because the only Database Ada programmer left the project on very short notice. The remaining team members were left to tackle a complex task with high required reliability while learning to use a complex language. The inexperienced Ada team did not use software tools and did not follow modern programming practices. However, the following question remains: just how much of the productivity differences do experience, tools and modern programming practices when combined with Ada explain?

Before we can answer that question we need to control for other known environmental influences. Some projects are more complex; others have a greater required reliability. If the database were large enough, we could estimate the influence of the environmental factors including the presence of Ada. Since there is not sufficient data, a second best solution is to use known estimates of the effort impact of the environmental factors. COCOMO provides estimates based on non-Ada projects. Therefore we can normalize for these factors using the COCOMO weights, and the remaining productivity variations between modules are likely to be related to the presence of Ada.

Assuming that

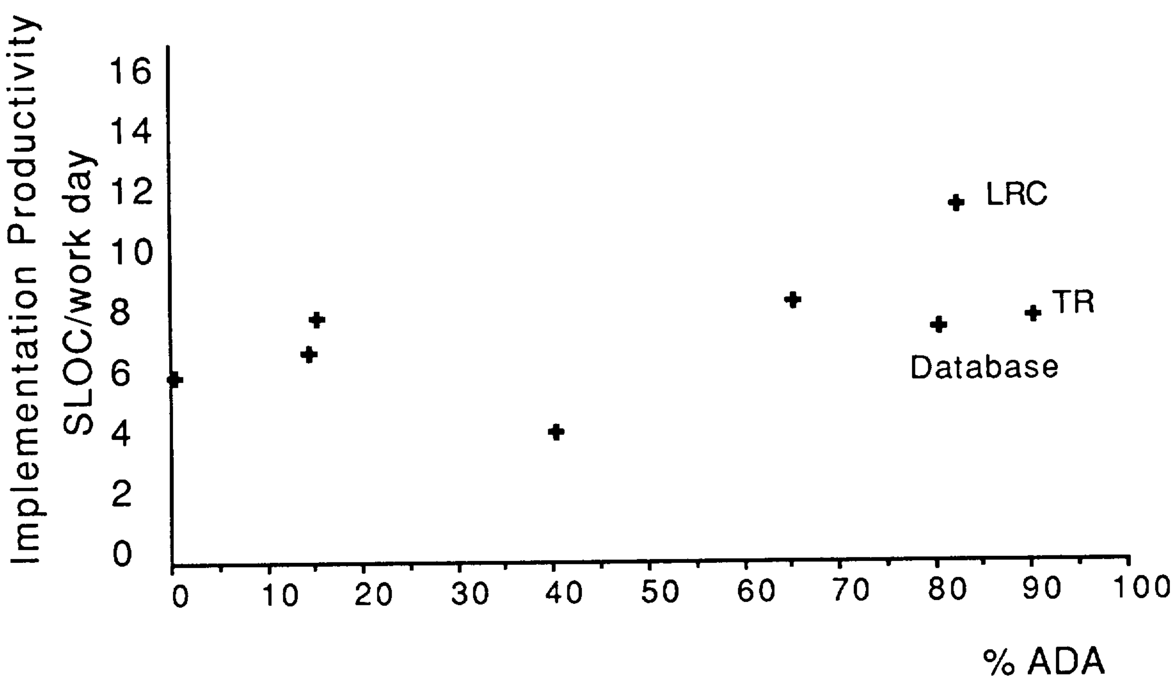
$$\text{Effort} = A \cdot L^{B} \cdot \text{EAF}$$

where L is executable source lines of code and EAF is the product of the cost drivers or environmental factors then adjusted effort is just Effort/EAF. Adjusted productivity then becomes

$$\text{ATOP} = [L/\text{Effort}] \cdot \text{EAF}.$$

Figure 2 displays the plot of adjusted productivity against % Ada . After adjusting for all the software development environmental factors except language experience the adjusted productivity values vary from 6.1 to 11.7 SLOC/work day. All but two subsystem adjusted productivities fall between 6.1 and 8.6 SLOC/work day.

Figure 2
Productivity Adjusted
Except for Language Experience



The average productivity for those module with less then 25% Ada is 6.9 SLOC/work day and for those modules with greater then 60% Ada it is 9.1 SLOC/work day. Based on a two-tailed t-test there is only a 10% probability that these represent the same distribution. Hence we can tentatively conclude that those projects with a high Ada content had a productivity 2.2 SLOC/work day higher then those with little or no Ada. Compared to the average productivity for the whole project this represents a 12% increase.

Within the group of modules with greater then 60% Ada the LRC module attained the highest productivity of 11.7 SLOC/work day which represents a 4.8 SLOC/work day increase or 25% improvement. The high productivity of Logistics is probably reflective of their being further up on the learning curve. Logistics did have one member who had the maximum possible Ada experience and substantial experience with object oriented programming. This suggests that three years of experience with Ada and an Ada programming environment might represent an important turning point. This point is further reinforced given that during the final release productivity increased to 27 SLOC/work day which is when those who started with about 1 year of Ada experience would have reached over three years of experience.

One other comparison that can be made is to compare the adjusted productivities between two similar projects one which uses Ada and one that does not use Ada. The comparison project used Pascal. These results are reported in Table 3. The comparison project is also a command and control task for the Air Force and even for the same contractor. The one major difference that cannot be controlled for is that Project 1 started out as a prototype but became an incrementally developed delivered system and project 2 was a prototype from beginning to end. After adjusting for differences in complexity and the lack of software tools the non-Ada project has a higher average adjusted productivity. Based on a two-tailed t-test there is only a 5% probability that these represent the same distribution.

The implication is that if you take away the tools and rules and adjust for differences in complexity and other environmental factors then the main impact of Ada as a language, without its tools lowers productivity when the programming staff has an average of one year experience. From the previous discussion we also suspect that once the experience level gets above three years then this difference will no longer be statistically significant.

Table 3

Average Productivity
(SLOC/work day)

	Total	Adjusted
Project 1 (Ada & C)	17.9	7.7
Project 2 (Pascal)	13.6	13.6

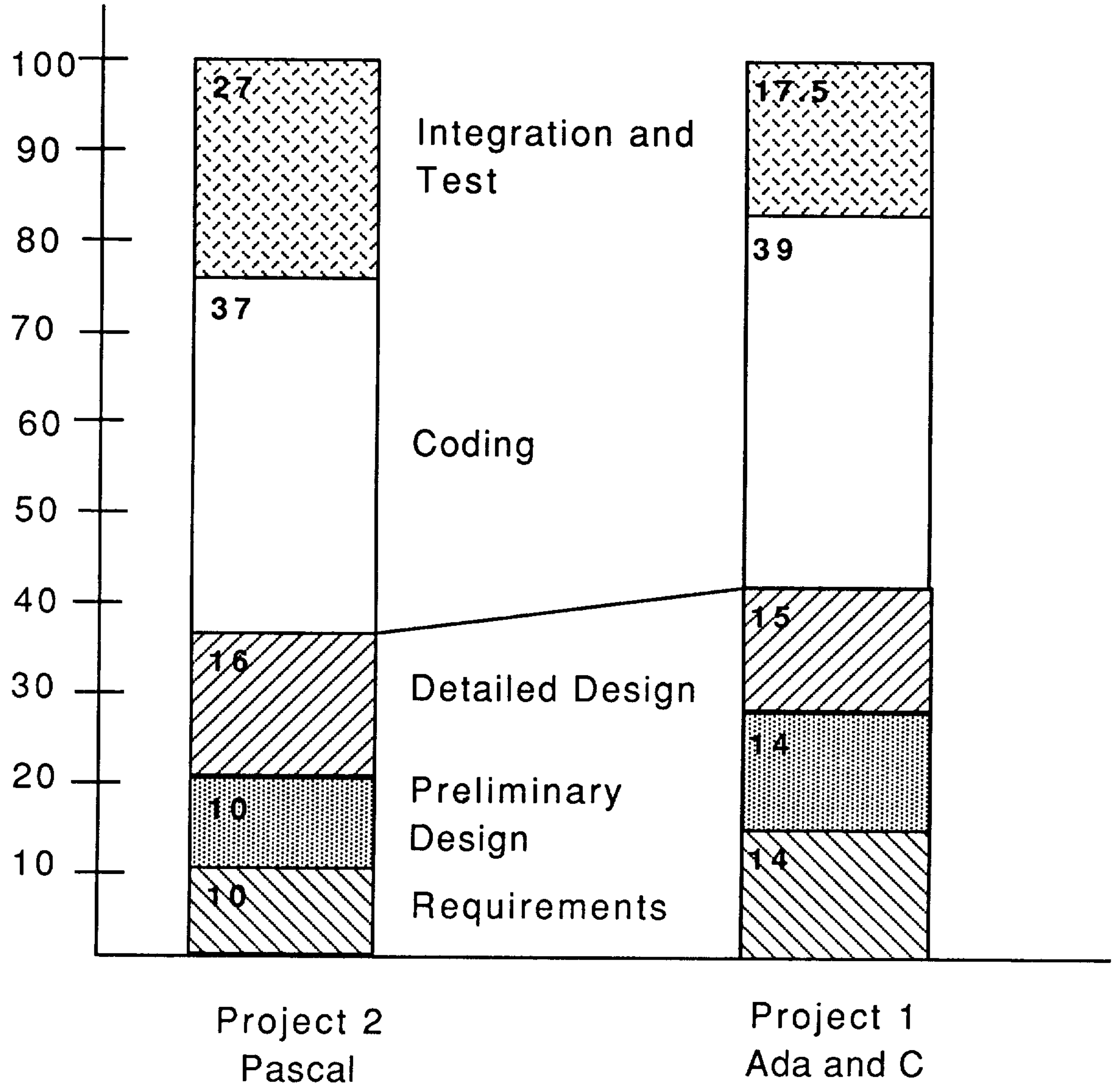
For this small sample the inference that can be drawn is that for experience of one year or less we can explain the majority of the observed variation in productivities by what we know about the impact of software tools, experience, etc on other languages. Software tools are important and a sophisticated programming environment will increase the productivity of any language. This interpretation must be discounted by the fact that Project 1 is a prototype and therefore the testing and integration phase plays a less significant role in determining development costs and it is here that one would expect Ada to have its most significant impact on development effort and productivity.

Ada and the Development Life Cycle

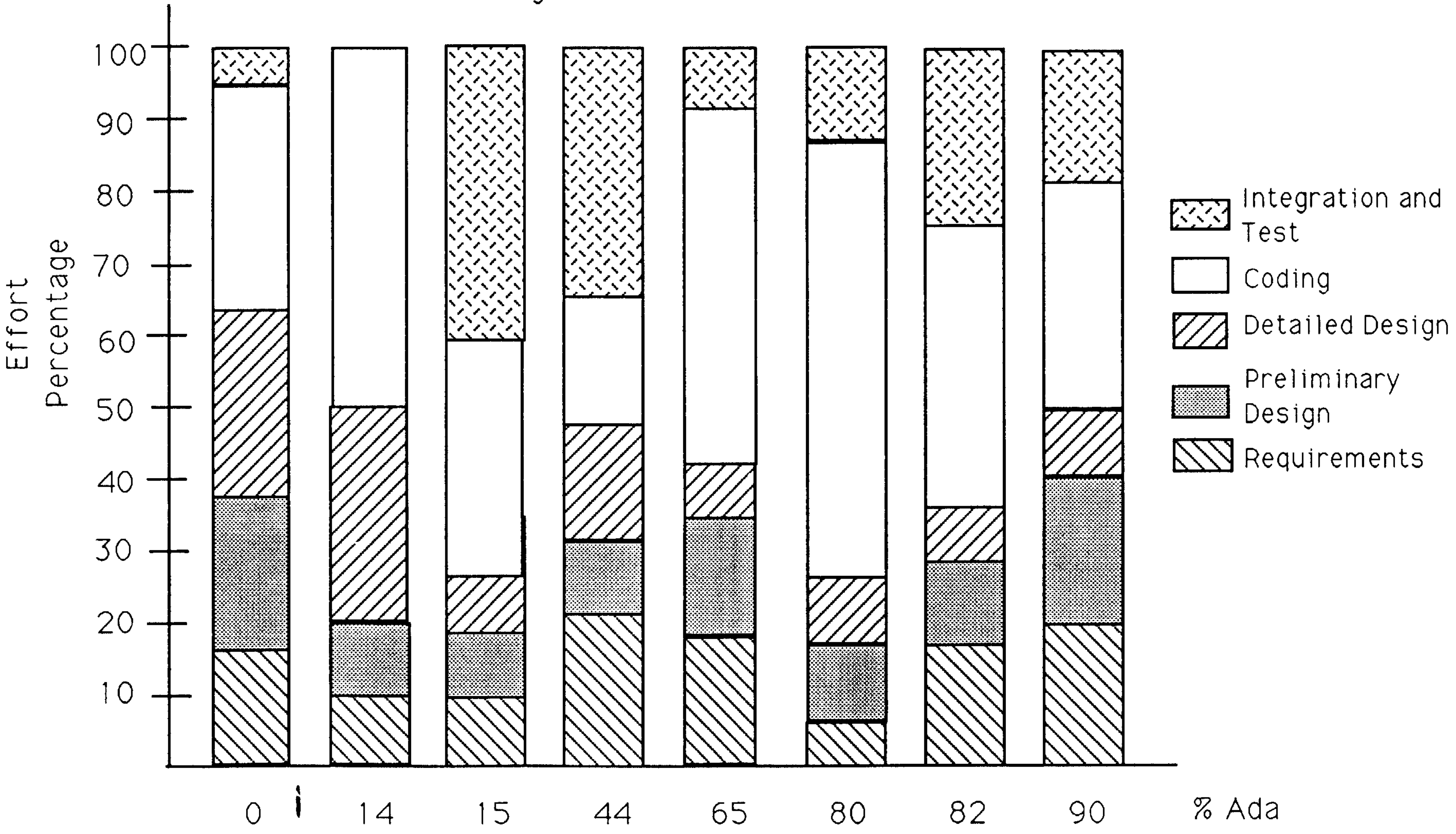
Previous studies have reported that Ada increases the effort in design, and decreases effort in the integration and test phase. One phase breakdown that has been reported is 50:33:17 for Ada and 40:38:22 for FORTRAN.[Royce, W. 1987] Comparing to Projects 1 and 2 again we can see to what extent this pattern holds up for prototypes. Figure 3 shows a phase breakdown for the whole project of 36:37:27 for Pascal and 43:39:18 for an Ada and C project. As expected, prototypes spend less time in design and more in coding. Furthermore the Ada prototype spends more time in design and less in testing then the non-Ada prototype.

While the effort by phase breakdown for the projects as a whole yields a consistent story the view from the module level does not. There does not appear to be any consistent pattern whatsoever.

Phase Distribution for
Command and Control Prototypes



Effort Distribution by Phase for Project 1 Modules



Conclusion

The data reflects the state of Project 1 before it actually became productized and therefore contains reduced effort figures for testing and documentation, which greatly increased during the final release. There is also not any data on maintenance costs. Therefore the two areas where Ada's strong type checking and compiler have an effect are not reflected. In addition there was no effort to make the code portable or reusable.

Any conclusions are tentative and should be treated as hypothesis for future research. As part of our continuing software costing analysis at JPL, two Ada projects and one Lisp project will be surveyed during 1989 which should make it possible to better isolate the impact of software tools and modern programming practices from other features of a language.

Given these caveats then our tentative conclusions are the following for Ada in a prototyping environment.

(1) Analyst and programmer experience in Ada of three years or more could increase Total Technical Productivity by 3-4 SLOC/day or a maximum of 25%.

(2) Technical experience and ability, modern programming practices and the use of software tools are very important in achieving high programmer productivity.

(3) For any language the combination of highly capable and experienced personnel, with the discipline of modern programming practices and a sophisticated programming environment should produce comparable levels of productivity to that observed for Ada in this study.

(4) Effort in the three major phases of the software lifecycle appears to shift such that time spent in design is increased and time spent in verification and test is decreased.

Bibliography

Boehm, B. 1981 Software Engineering Economics, Prentice Hall.

Economics Group 1989 NASA Historical Database, JPL/Caltech, January 1989.

Reifer, D. 1988 Softcost-Ada: An Update, Fourth Annual COCOMO Users' Group Meeting Workshop, Pittsburgh, Pa., Nov. 2-3, 1988

Royce, W. 1987 Estimating Ada Software Development Costs for C^3 Systems, TRW Defense Systems Group, Preprint.

SORCE/Economics Group 1988 Software Productivity Analysis Database, JPL/Caltech, 1988.

THE VIEWGRAPH MATERIALS

FOR THE

J. HIHN PRESENTATION FOLLOW

Ada Productivity Analysis: A Case Study

Jairus M. Hihn
Hamid Habib-agahi
Shan Malhotra

JPL

Jet Propulsion Laboratory
California Institute of Technology

December 13, 1988

Outline

Project Overview

Command and Control Projects

Ada Claims

Analysis

Conclusions

Software Database Description

- *Size* - executable source lines of code (SLOC) without comments
  adjusted for inherited and modified code

- *Cost* - work months assuming 19 working days per month
  technical effort from interviews and implementation effort from task management

- *Effort breakdown by phase*

- *Development mode*

- **Product Attributes**
  Required Reliability
  Software Complexity
  Database Size
  Development Language

- **Computer Attributes**
  *Time Constraints**
  *Storage Constraints**
  *Virtual Machine Volatility**
  Turnaround Time

- **Personnel Attributes**
  Analyst Capability
  Analyst Experience
  Programmer Capability
  Language Experience
  Virtual Machine Experience

- **Project Attributes**
  Software Tools
  Modern Programming Practices
  *Schedule**
  Requirements Volatility

Average Productivities

NASA Historical Database, 10 projects, 100 modules

Flight S/W averaged approximately 1.5-3.5 SLOC/work day

Ground S/W averaged approximately 7-10 SLOC/work day

JPL Software Database, 4 projects, 39 modules

Project	SLOC/work day
1	17.9
2	13.6
3	15.4
4	6.5

Project 1
Command and Control

Development Data

Module	Size (KSLOC)	Technical Effort (Work Months)	Implementation Effort (Work Months)	Technical Productivity (SLOC/work day)	Implementation Productivity (SLOC/work day)
Application Software					
DOC	72	118	207	32	18
DOO	115	140	245	43	25
LRC	45	28	49	84	48
CAT	23	36	63	34	20
TR	70	60	105	61	35
System Software					
Graphics	20	72	145	15	8.3
Common	110	258	453	22	13
Database	37	110	193	17.7	11
Total	492	822	1,460	31.5	17.9

Project 1
Development Environment

Ada

Prototype

- Documentation
- Testing

Project combined known technologies and techniques

Excellent communication existed between users and developers

Personnel were well matched to tasks

Project 1
Development Environment

Product Attributes	Low to Nominal
Computer Attributes	Low to Nominal
Personnel Attributes	High
Project Attributes	High

Ada Claims

Ada will lower life cycle costs in long term projects

- Maintenance costs

Ada will/may lower development costs

- Integration and test
- Coding

Caveats

The results should be viewed as hypotheses for future research since

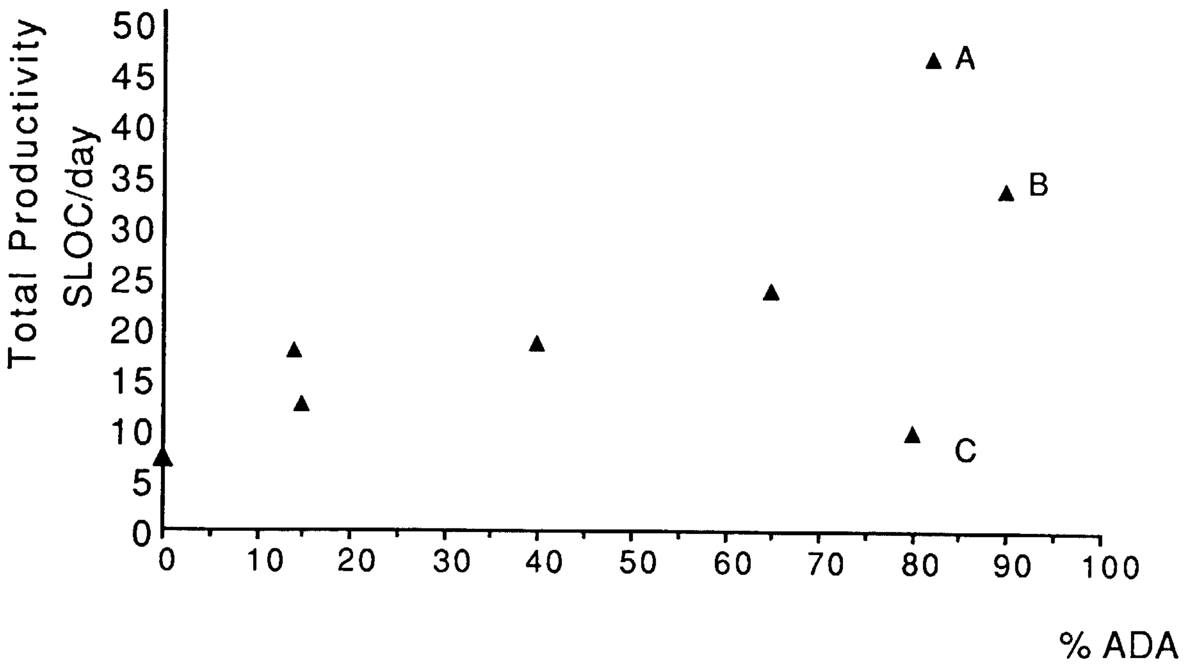
the sample size is small

care must be taken when comparing “lines of code”
from different languages

effort was not expended to support reuse or portability

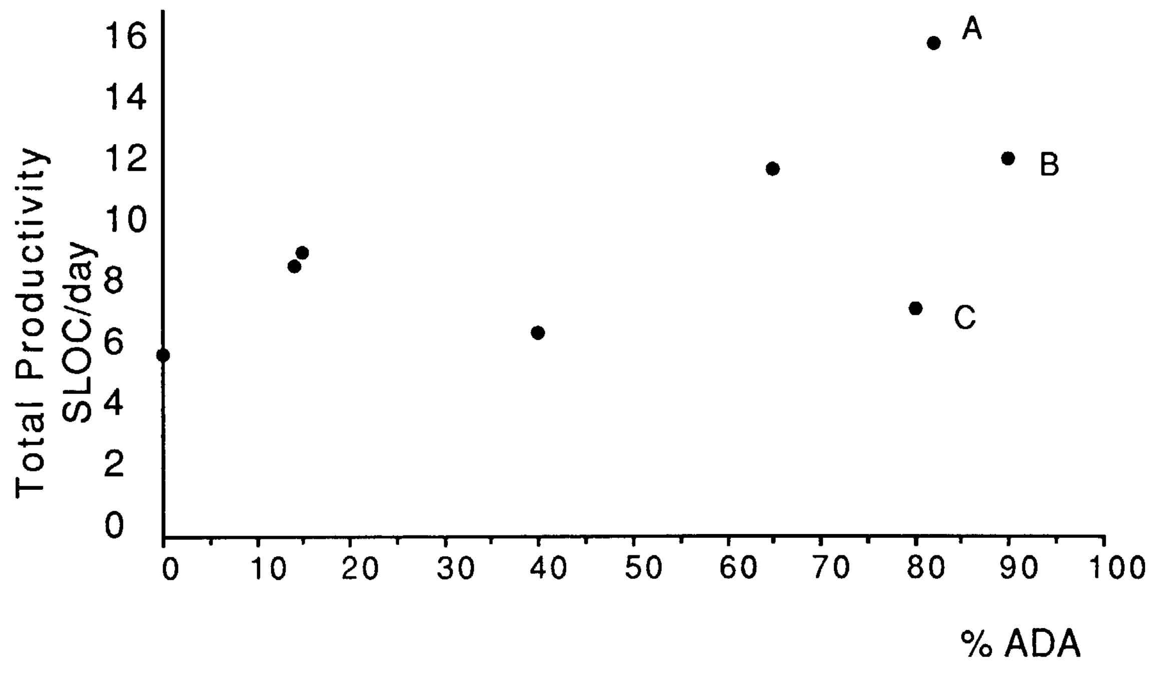
the data represents a prototype and Ada is designed
for production systems

Impact of Ada on Productivity ?

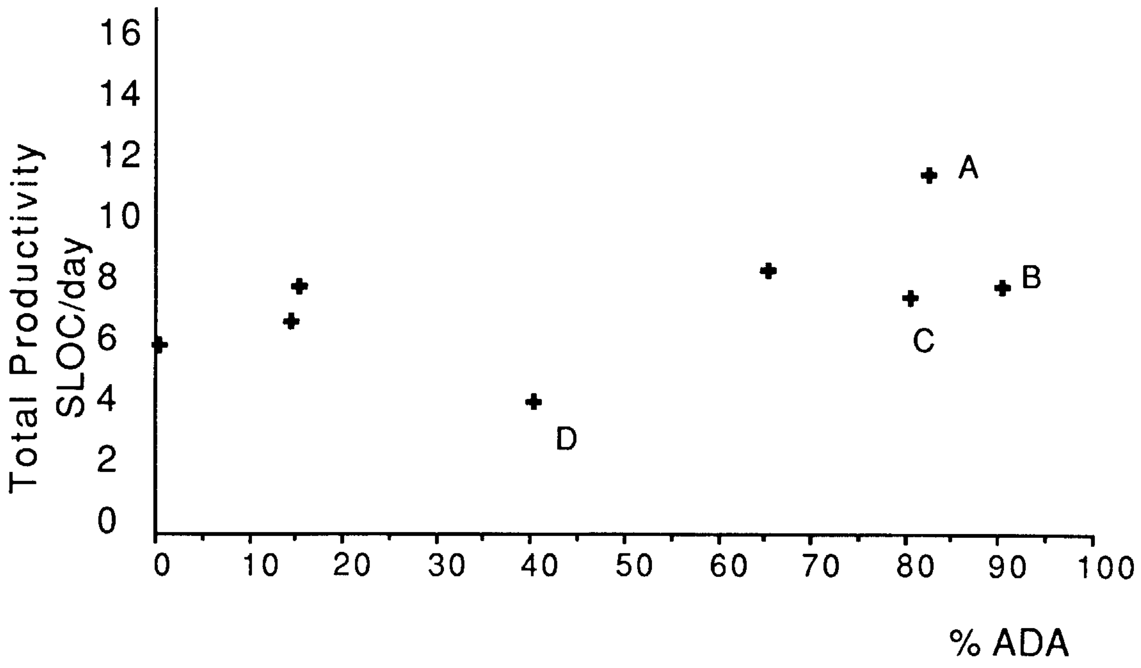


There appears to be a general increase in productivity as the % of Ada increases in a module.

Productivity Partially Adjusted
Except for Software Tools, Modern Programing Practices
and Language Experience



Productivity Adjusted
Except for Language Experience



Project 1 compared to Project 2
Command and Control

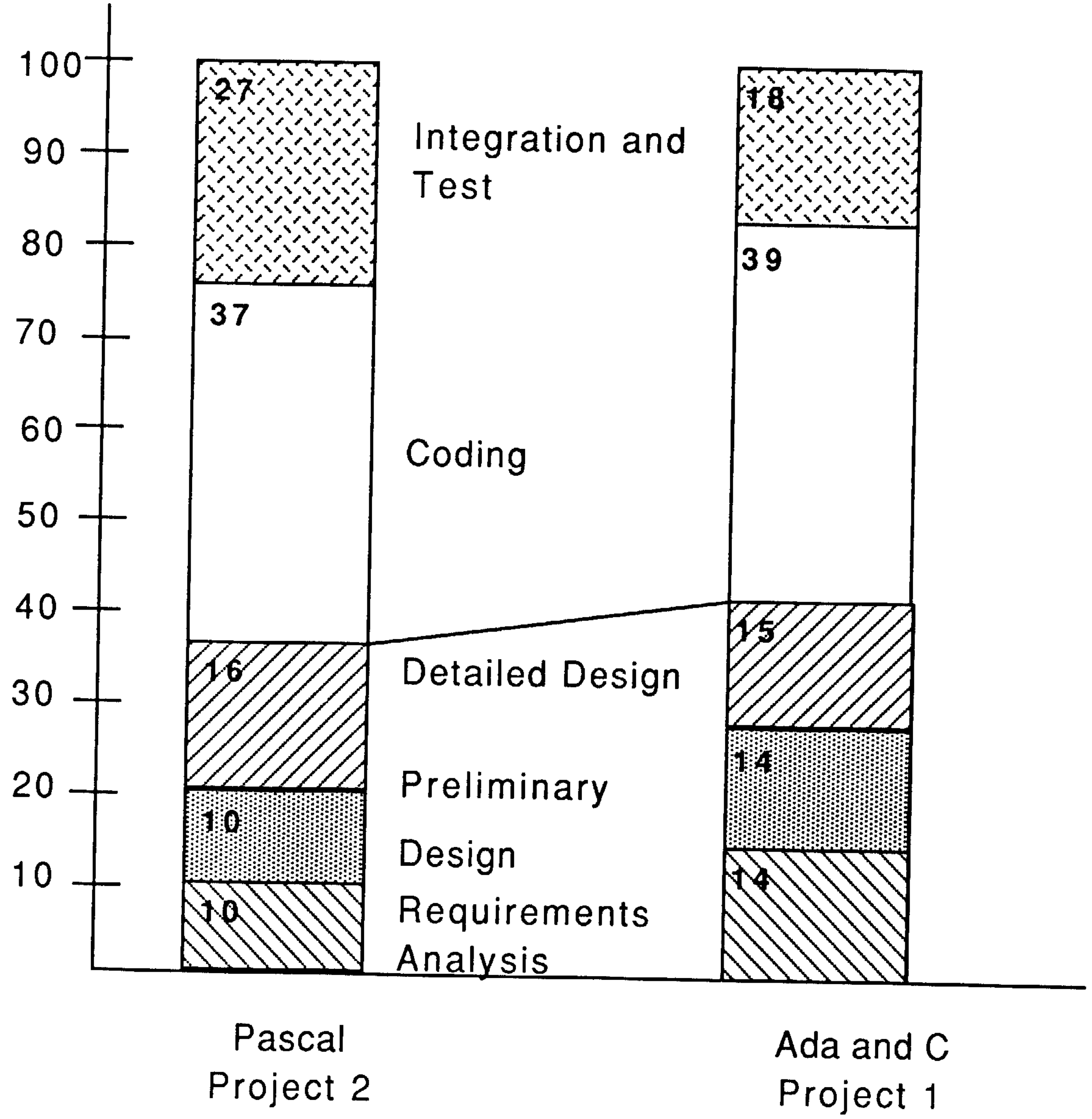
The average productivity for project 1 (Ada and C)

Total productivity	= 17.9 SLOC/work day
Adjusted productivity	= 7.4 SLOC/work day

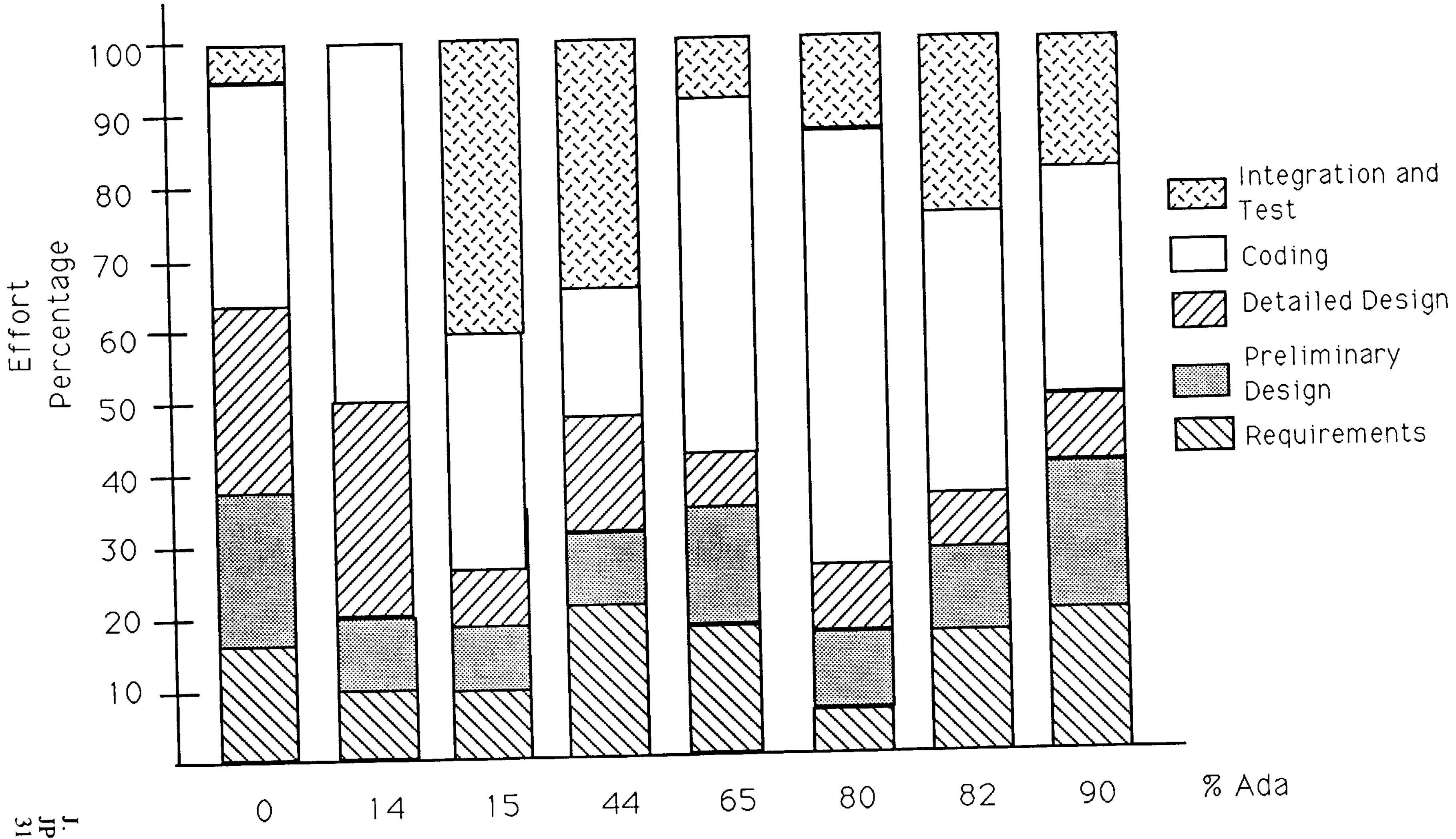
The average productivity for project 2 (Pascal)

Total productivity	= 13.6 SLOC/work day
Adjusted productivity	= 13.5 SLOC/work day

Phase Distribution for Command and Control Prototypes Project Level



Effort Distribution by Phase for Project 1 Modules



Conclusions

These results are reflective of a new project which has hired very capable people with extensive general experience but on average 1 year experience in Ada.

10-25% increase possible with Language experience > 3 years

Similar productivity gains should be possible with other languages

- Programming environments
- Modern programming practices

Impact of development phases

- Increased design and requirements
- Decreased integration and test