IMPROVING SPEED AND PRODUCTIVITY OF SOFTWARE DEVELOPMENT: A SURVEY OF EUROPEAN SOFTWARE DEVELOPERS

by

J. BLACKBURN*
G. SCUDDER**
and
L.N. VAN WASSENHOVE†

95/88/TM

* Professor, at Owen Graduate School of Management, Vanderbilt University, Nashville, Tennessee 37203, USA.

** Professor, at Owen Graduate School of Management, Vanderbilt University, Nashville, Tennessee 37203, USA.

† Professor of Operations Management and Operations Research at INSEAD, Boulevard de Constance, Fontainebleau 77305 Cedex, France.

Printed at INSEAD, Fontainebleau, France

Improving Speed and Productivity of Software Development: A Survey of European Software Developers

Working Paper # 95-31

October, 1995

Joseph Blackburn
Owen Graduate School of Management
Vanderbilt University

Gary Scudder
Owen Graduate School of Management
Vanderbilt University

Luk N. Van Wassenhove
Technology Management
INSEAD
Fontainebleau, France

ABSTRACT

Improving Speed and Productivity of Software Development: A Survey of European Software Developers

Time is an essential measure of performance in software development because time delays tend to fall directly to the bottom line. To address this issue, this research seeks to distinguish time-based software development practices: those managerial actions that result in faster development speed. This study is based upon a survey of software management practices in Western Europe and builds upon an earlier study we carried out in the U.S. and Japan. We measure the extent to which managers in the U.S., Japan and Europe differ in their management of software projects and also determine the tools, technology and practices that separate fast and slow developers.

I. Introduction

Time is an essential measure of performance in software development. For commercial software, such as spreadsheets and operating systems, market share erodes rapidly with time because customers will not wait for a delayed product when alternatives are readily available. In consumer electronics or telecommunications equipment, where software is designed in parallel with hardware, project delays often result in market entry with a less attractive product. Software tends to lie on the critical path and when the development of software features misses the product launch date, the product enters the market without the features. Time delays in software tend to fall right to the bottom line.

When time is a critical performance metric, most software organizations fare poorly because software projects are routinely late and over budget [1]. The penalties of being late in a competitive market can be devastating: Ashton-Tate, a PC database software leader, lost its command of the market because it took too long to complete a new version of dBase. After the CEO departed and the company was sold, the new owners underestimated the time required to finish the new version, ultimately scrapped it and started over [2]. More recently, IBM's release of its PowerPC-based machines was late to the market due to software delays [3]. In addition to the penalties meted out by the market, project delays create inescapable deadline pressure, which often results in defective products rushed out the door and, subsequently, unhappy customers. Dissatisfied customers can diminish the sales of current products through word-of-mouth and are unlikely to purchase future product releases.

This research addresses these problems by discerning the management practices that result in faster software development, which we call time-based software development. For other forms of product development where this theme is well established, a number of principles have been advanced for managing the time-to-market process: determining precise customer specifications up front, concurrent engineering with co-located, cross-functional

teams, etc. (see for example, Blackburn [4], Stalk and Hout [5] and Smith and Reinertsen [6]). Yet software is often viewed, at least by software managers, as different. They claim that writing software is an art, not a science, and must be managed as a craft, if it can be managed at all.

An objective of this research is to measure the extent to which practices, such as concurrent engineering, found to be successful in reducing development time for non-software products are applicable to the management of the software process. Although there may be significant differences between software and hardware, we contend that software, as an industry, has become too large, too important and too tightly linked with hardware to be treated as an unmanageable problem child or an 'add-on' to be programmed after hardware has been designed.

The research described here is based on a survey of software management practices in Western Europe. It follows and builds upon an earlier study of software managers that we carried out in the U.S. and Japan in 1992 and 1993 ([7],[8],[9],) and field interviews conducted in 1992 with product development managers in Europe [10]. A study which considers issues of productivity and quality in software was carried out by Curtis, Krasner and Iscoe [11] and one of the authors (van Wassenhove) has participated in a study of European software productivity within a specific application area [12].

Our European survey addressed three important questions concerning managerial actions to improve software processes:

(1) To what extent do managers in the U.S., Japan and Europe differ in their direction of software development activities?

(2) What management actions accelerate software development speed?

(3) What management actions support higher productivity?

The distinction we make in questions 2 and 3 between the rate of change in development speed and productivity is an important one. Development speed and productivity are not the same because low productivity organizations can be quicker to market by throwing more human resources-- armies of programmers -- at the project. However, research by Brooks

[13] suggests that such efforts are futile because throwing more bodies at a project, particularly in the latter stages, actually can extend the development time. There are many factors affecting productivity that are beyond a given manager's (or even the firm's) control, such as complexity of task, project size, and whether hardware design is also involved. Although a firm may appear productive by posting high lines-of-code (LOC) productivity numbers, development cycle times may also be static and not changing rapidly. A firm with more daunting problems and with apparent low productivity may, in fact, be ramping up their speed dramatically. In this research we will attempt to understand the extent to which similar management practices enhance speed and productivity in software. Through comparisons with our previous research, we also determine if there exist significant differences in software management by global region, what constitutes best practice, and where it is conducted.

To initiate this survey, we used a mailing list of European managers from INSEAD in Fontainebleau, France. From that list the names of 623 managers in software-related industries were selected from the following SICs: 3579, 3600, 3700, 3720, 3800,4000, 4800, 7370, 8900 and 8910. Only large firms were selected; for firms outside of France, we screened out all firms with fewer than 200 employees and less than $200 million in annual sales; since the list contained a preponderance of French firms, the screening levels for French firms were somewhat higher. As a result, about two-thirds of the sample were drawn roughly equally from firms in France, the UK and Germany, with the remainder distributed among other countries in the EU. The total response rate was 27.3 per cent or 170 responses. Of these, 72 were replies declining to participate, leaving 98 or 15.7% usable responses.

The survey first asked respondents to describe a recently-completed software project: the type of project, language used, procedures to manage and monitor the process, and performance measures. We then asked them to supply quantitative information about the project in five areas:

(1) project size and productivity over time;

(2) allocation of time, effort and team size among project phases;

(3) degree of newness of project (from minor revision to completely new);

(4) the effectiveness of different tools and techniques for time compressing the development process;

(5) stages of process in which time reductions have been achieved.

II. Global Comparisons of Software Management Practices

Comparing sample results from the three global regions, we were struck more by their similarities than differences. On average, firms in Japan, the U.S. and Europe allocate time to the various stages of a project and effort in terms of man-months per stage in roughly the same proportions. Figure 1 presents a region-by-region comparison of the percentage of development cycle time spent in each phase of the project. In all the regions the firms devote about 15% of the elapsed time in the customer requirements and the planning and specifications stage. The only significant difference in time allocation (at the .05 level) is between Japan and Europe in the coding and implementation stage where the European firms devote an average 29% of the time and the Japanese firms 22%. Figure 2 shows how effort (in % of total project man-months) is allocated across the different software development phases.

Comparison of Figure 2 with Figure 1 shows that the percentage of effort in the early project stages is lower than the percentage of elapsed time for those same stages: firms allocate fewer resources to the project in the early stages, but the effort grows steadily from planning and specifications through coding and implementation. Japanese firms devote more effort than their European counterparts in the planning/specification process and less effort in coding/impementation (both differences are significant at the .05 level). This result echoes an observation frequently made about Japanese product development practices: their teams spend more time in planning and, consequently, less in execution of the detailed design work.

In the survey we asked respondents to consider a recently-completed project and to estimate the duration of the project if it had been undertaken five years earlier. That is, we asked for the respondents' estimates of their rate of change in product development speed. Most indicated that they were now faster; a similar project would have taken longer if attempted five years previously. The average percentage reduction in development time

reported in the sample was 27%. Although this estimate is based on perceptions and is clearly subject to bias on the part of respondents wanting to show improvement, the value is remarkably consistent with our earlier surveys carried out in the U.S./Japan, in which the average improvement in development time was 28%. In our analysis consistency is important because bias, per se, is not a major concern: we are interested in relative, not absolute, levels of improvement. That is, we partition the sample into firms that are on a fast rate of change in development speed and those that are slower and then isolate significant differences between the samples. So we only require that the firms be roughly consistent in their estimates.

We also asked respondents, "To what extent were the following factors useful in reducing the overall software development time for the project?" The eleven factors were chosen from those that have been identified in the literature:

1. The use of prototyping: prototypes generate customer input by demonstrating to the user how the proposed software will work;

2. Better customer specifications initially: time spent learning what the customer wants or needs up front should reduce the frequency of specification changes later in the project;

3. The use of CASE tools: these tools are promoted in the software engineering literature as productivity-enhancing technology which shrinks development time;

4. Concurrent development of stages or modules: parallel, or overlapping, design activities should reduce development time;

5. Less rework or recoding: improved first-pass quality should compress project completion time;

6. Improved project team management: in development, cross-functional teams tend to outperform the functional, "over-the-wall" approach;

7. Better testing strategies: software testing developed prior to, and in parallel with, product design should minimize the rework required to fix problems discovered late in the project and in the field;

8. Reuse of code or modules: reusing previously-tested code improves productivity, and thus, development time, by reducing the need to create new code;

9. Changes in module size and/or linkages: smaller modules or more standard interfaces speed coding and testing by supporting parallel development;

10. Improvements in communication between team members: high band-width information exchange among a cross-functional team supports overlapping activities and prevents errors due to misunderstanding;

11. Better programmers or software engineers: hiring the best people and supporting them with training enhances productivity and speed.

We asked respondents to rank these eleven factors on a 1-5 scale from 1=Not at all to 5=Very Helpful. In addition, the participants were asked: To the extent that the factors were useful in reducing development time, in which stages of the project were the reductions achieved? If you had developed the same software product five years ago, how long would the project have taken?

The averages of the importance assessments of the eleven factors, shown in Figure 3, compare the results of the recent European survey with those from earlier surveys in the U.S. and Japan. Although the project management factors receive roughly the same weightings in terms of their effect on reducing product development times, there are some significant differences. Reuse is viewed as more important in the US sample; the Japanese firms place a lower average importance weight on prototyping and customer specifications.

"People factors" seem to dominate "tools and techniques." In the European sample, the three highest-rated factors are better customer specifications, communications and better programmers. The lowest values are given to changing module size and/or linkages and surprisingly low weightings are associated with CASE tools and technology in all three samples (the US firms rate CASE tools significantly lower in importance than European firms). Although much of the literature and popular press extols the importance of new tools and technology for enhancing software development, the perception of users is that these tools have a relatively minor effect on product development time. In interviews managers suggested that problem complexity and the need to deal with people issues were overwhelming their ability to use the tools effectively

Averages, by obscuring differences between high and low performers, only tell part of the story. But which project management factors really make a difference in development speed

and productivity? What matters is the activities that separate the firms that are accelerating their development time from the rest of the pack. What different things are done by the fast movers? The analysis to isolate the drivers of change is described in the next two sections.

III. Accelerating Product Development Speed

In this section we seek to identify the specific activities that a firm should stress to shorten their development cycles. To do this, we examine the correlations between the percentage reduction in development time (over the past five years) and the actions that software managers take to reduce cycle time. The results indicate that in the allocation of time and effort, project managers should use a similar approach for productivity and cycle time reduction. The strongest implication from this analysis is the importance of customer requirements. This theme of "get to know what your customers want" is stated often in the new product development literature, and these data provide solid evidence that the fast-track software firms are following the theme.

A. Allocation of Effort and Time across Project Stages

The firms improving their development speed at the fastest rate actually spend more elapsed time and more effort in the customer requirements stage of the project-- that is, determining what the customer wants in the software before proceeding into high-level planning, designing and coding. These firms spend significantly less time in the testing and integration stage of development.

We partitioned the sample into two groups based on development speed: fast firms (Group F) had improved their development speed by greater than 25% (over the past five years) and slower firms (Group S) had development speed improvements of 25% or less. Figure 4 shows how these two groups of firms differ in the percentage allocation of time and effort across development stages. For example, the faster firms devote an average 18% of their development cycle time and 14% of their man-hours effort to determining customer requirements. The slower firms expend about one-half the effort in determining customer requirements. (These

differences are significant at the .05 level). On the other hand, the faster firms spend less time and effort on average in all of the other stages of development and significantly less time in the final, testing/integration stage.

Table 1 presents the correlations between reduction in development speed and time and effort spent in the various stages and their significance levels. These correlations confirm what the sample averages show: high improvement rates in development speed tend to be positively correlated with the more time and effort in the customer requirements and negatively correlated with time spent in the testing/integration stage. Taken together, these results suggest, as a policy measure, that more time spent up front defining customer requirements forestalls the need for testing later.

B. Effects of Team Size

Does team size affect development speed? Figure 5 displays the average team sizes by project stage and maximum team size for the fast and slow development groups. This figure shows that, except for the customer requirements stage, the faster firms tend to have smaller teams. These results support the observation often made in the new product development literature that small, "tiger teams" tend to be more effective. Unlike the present situation, however, these assertions about small team size are rarely supported by other than anecdotal evidence.

The larger team size in the customer requirements stage by the faster firms provides additional evidence that the initial stage is a critical part of the process. Devoting more resources in the early stages of a software development project to learning what customers want should be viewed as an investment, not just a cost.

C. Analysis of the Eleven Project Management Factors

The correlation coefficients (shown in Table 1) between the importance of the eleven project management factors for reducing development time and the reported percent decrease in development time provide additional insight into the actions taken in faster firms. These

correlations indicate that the firms with large reported decreases in development time have accomplished that through an increased dependence on prototyping, better programmers and less rework. Note that large improvements in development time are (weakly) negatively correlated with better testing strategies. This suggests that the firms reporting the greatest improvements in development time perhaps have fewer problems with rework and therefore have less dependence on testing; the firms reporting benefits from testing strategies may be relying on testing to fix problems.

Figure 3 and the correlations in Table 1 suggest that CASE tools are perceived to have an insignificant effect on reduction in development time. Since much has been written in the literature and the popular press about CASE tools and technology and their effect on productivity, we expected the correlation to be strong, positive and significant. The same could be said for reuse. Neither of these appear to be as powerful a driver for reducing project cycle time as prototyping, reducing rework and better programmers. If CASE tools are contributing to reduced cycle times, then the effect is either overshadowed by other factors or is obscured by the increasing complexity of the development task.

Taken together with the accumulated evidence on the importance of the customer requirements activity, the correlations of percent reduction in development time provide persuasive evidence that, for faster cycle times, it is critical to "do it right the first time." In software, prototyping is a useful tool for the process of hammering out a clear set of specifications with users by getting preliminary feedback on "features and feel" without having a complete design. Less rework is probably more a consequence of getting project specifications right than a cause. One plausible interpretation for what respondents mean by better programmers is people who get the job done right.

Talented people are essential to a fast development process. These results emphasize a fact of life in development: virtually no amount of management technique and team organization can overcome a lack of talented designers and coders. This is not a new observation, but is a recurring theme in the literature on innovation, research and development. The lesson for managers is that, since these talented people are so important to

the time-to-market process, the firm should make a special effort to identify, reward and make heroes of their best people. Making most effective use of development talent must be a key concern for software managers.

IV. Software Productivity Drivers

The software research literature teems with attempts to explain, model and predict software productivity, to find reliable metrics and to account for differences due to language and type of application (see Maxwell, et al. [12] for a recent study and a review of prior literature). Our purpose is not to develop another model to predict software productivity; our dual purposes are

(1) to determine the management practices that support higher productivity by examining the differences in procedures between low and high productivity firms;

(2) to provide supporting evidence for the conclusions of the prior section about factors important in reducing cycle time.

Although the results in the prior section are based on the respondents' perceptions of cycle time reductions, this section provides direct, quantitative measures of productivity; this provides a validity check on whether similar actions support both productivity and cycle time reductions.

In examining productivity, we used lines of code (LOC) as a measure of program size simply because it is the most common measure, one with which most of our survey respondents are comfortable and could be used to describe the size of their project. [Although function points have been suggested by Jones [14] as a better measure, only one respondent reported program size in terms of function points.] In the European sample, of the 98 respondents, 59 were able to provide both project effort in man-months duration by stage and program size in terms of lines of code, so that we could compute Lines-of-Code per Total Man Months (LOC/TMM) and Lines-of-Code per Man Month of Coding Effort (LOC/MMC) as measures of productivity. Not surprisingly, the correlations between these productivity measures and different project descriptors were quite similar because LOC/TMM and LOC/MMC tend to be highly correlated with each other (in our sample, the correlation was

0.66) and project management factors that improve coding productivity tend to improve total project productivity. This analysis will focus on LOC/MMC; the conclusions reached also apply to LOC/TMM productivity because the results are similar.

The reported productivity values exhibited a larger variance than anticipated, due to the presence of several large outliers. Four firms reported productivity values more than 3 standard deviations above the mean. When these could not be verified, they were eliminated from the productivity analysis in this section because they appeared to be either mistakes or miscalculations on the part of the respondents. The remainder of the sample had a mean productivity of 3041 LOC/MMC with a standard deviation of 3400 (the mean LOC/TMM was 1124). The high variation in productivity in the sample is not surprising given the diversity of software projects: the sample includes small business applications, large telecommunications projects, and software designed in parallel with hardware design. Programs ranged in size from 1200 lines of code up to 6 million. In addition, there are twenty-seven different programming languages and the projects were carried out in twelve different countries in Europe.

But what explains differences in productivity across firms? To uncover the project management factors driving productivity, we computed correlations between LOC/MMC and the various project factors elicited in the survey. In the sections that follow, we analyze the effects of program size, language, allocation of time and effort, and the eleven project management factors introduced earlier.

A. Productivity and Development Time

Are the firms with faster development cycles more productive? If firms are crunching development time by throwing human resources at the project, then productivity would diminish. On the contrary, the faster firms tend to be more productive. Figure 6 displays the average productivity measures for the fast firms (Group F) and slow (Group S), and the productivity of the fast firms is significantly higher (at the 0.10 level). This suggests that the actions that these firms take to reduce development time also support productivity gains. Time-to-market targets and lower development cost are congruent objectives.

B. Project Characteristics and Productivity

Table 1 displays the statistical correlations between LOC/MMC productivity and different project descriptors relating to the size and complexity of the project: project duration, size in terms of lines of code, number of modules, newness of the project and team size during the project. The correlation between productivity and project duration is negative and significant at the 0.10 level: in this sample, productivity decreases with project duration. A similar result was obtained in an earlier study conducted by one of the authors for the European Space Agency [12].

The relationship is less clear between productivity and project size in terms of lines of code. As given in Table 1, the correlations between project size (as measured by lines of code) and productivity are positive. This result, while in accord with the findings in [12], does not agree with that found in some other studies ([15], [16], for example). Some authors argue that software productivity should diminish with project size because communication problems and other forms of project complexity tend to increase with the size of the project; increasing complexity creates efficiency losses resulting in lower productivity. Our data do not necessarily refute that argument because each of our sample points represents a different firm, and the firms in our sample with large projects may have better procedures in place for managing software projects than the small ones. For a better test of the size/productivity hypothesis, a sample of projects of different sizes within a single firm should be analyzed.

Survey respondents assessed project newness by ticking a scale ranging from 0% to 100% (completely new). The correlation between newness and productivity and between newness and percent reduction in development time were not significant.

C. Team Size

What is the effect of team size on productivity? Table 1 shows the correlation between LOC/MMC and team size in different stages of the development process. Our results on team size tend to support the conventional view that larger teams diminish productivity because of inefficiencies created by the difficulty of communicating within a large number of people.

Brooks [13] has argued that communication demands increase in proportion to the square of the size of the team. The correlation between maximum team size and lines-of-code productivity is weakly negative. There is, for example, a significant negative correlation with the team size in testing: firms that must do a lot of testing are less productive. They are spending an above average amount of time looking for, finding and fixing errors. The only slightly positive correlation between LOC/MMC and team size is in the customer requirements stage. To risk a generalization, the high productivity firms tend to have a larger team devoted to determining customer requirements (which, if done right, may make coding more productive and less testing and correction necessary). Further evidence in support of this assertion about the importance of customer requirements is provided in subsequent analysis.

D. Allocation of Time in Project Stages

The correlation between LOC/MMC and the percentage of project time spent in the various stages is displayed in Table 1. Here the only stage with a positive correlation, one that is strong and significant, is the time spent on customer requirements. Correlations between LOC/MMC productivity and the allocation of effort across stages provides additional confirmation of the importance of the initial customer requirements stage: the most productive firms allocate significantly more effort to this stage of the process.

Viewed together, the correlation with team size annd allocation of time in stages tell an interesting story about the importance of the initial stages of a project to coding productivity. Note in Table 1 that, while most of the correlations with team size are negative, but not significant, the only positive value is the size of the team working on customer requirements. Our conclusion from this is that firms allocating more effort up front-- in determining customer requirements and in high-level planning-- are getting a payoff in higher productivity. The link between time spent in the early stages of the project and time saved in coding may seem tenuous, but the evidence suggests that how the early stages are managed is a critical determinant of coding productivity. The lesson is: spend more time up front to save time and effort later on.

E. Language

In the European survey the choice of programming language does not correlate with productivity, either in LOC/MMC or LOC/TMM. The languages used ranged from low-level assembler code to high-level languages such as C++. Of these, C was the most prevalent, used by 25 (out of 96) of the respondents. After scaling the language by level, the correlations with productivity were slightly negative, but not significant. This was somewhat unexpected because, in an earlier study by one of the co-authors [12], a significant correlation was found between language and productivity, with higher levels of language (ADA and PASCAL) yielding higher lines-of-code productivity. However, this earlier study did not include projects with higher level languages such as C and C++ (as was the case in the current study). Because of the preponderance of these higher level languages in our survey, we may be seeing the effect described by Jones [14]: "when programs are written in higher level languages, their apparent productivity expressed in source code per time unit is lower than for similar applications written in lower level languages."

V. Conclusions

Software is an enigma: it pervades our lives and our products in increasing proportions, yet we struggle to meet project deadlines and to remain within budget. Many practitioners argue that software is an arcane art and resist attempts to structure and manage the process. Skeptics also suggest that attempts to develop guidelines would be overwhelmed by differences among projects, firms, and cultures.

This research suggests that the skeptics are wrong. Globally, firms appear to be remarkably similar in the way they structure the software process. Despite the ink devoted to Japanese "software factories", the Japanese firms, on average, organize their software efforts in ways that mirror U.S. and European firms. Productivity and rates of improvement do not seem

to be "culture-dependent" -- they are at comparable levels in each of these regions. World-wide similarities in management of the process are more apparent than the differences.

Important differences emerge, however, when firms are segmented by development performance instead of by country, or culture. The firms on a "fast track" in improving time-to-market manage their processes in significantly different ways and with better results than the slower firms. Moreover, similar actions support both speed and productivity, so managers do not have to treat these two performance measures as a tradeoff.

The development manager's most powerful lever is the initial activity of ascertaining customer requirements. Fast developers devote more time in the early stages of the project to learning what customers want in a software product and to shaping the specifications to meet those needs. Not only do the fastest firms spend more time and effort on user (or customer) requirements, but the firms with highest productivity follow the same actions (of course, many firms fall into both groups). Prototyping, which provides early feedback from customers on product features and user interfaces, is a technique more widely-used by the fast developers than the slow. "Getting closer to the customer" has been such a frequent admonition in the business press that it has become a trite cliche. However, this research confirms that, in software development, the advice about learning the requirements of the user is more than a cliche; it is a way of becoming faster and more productive than your competitors.

Time and effort spent on prototyping and other techniques to fine-tune customer requirements pay off in a shorter development cycle. Our results strongly suggest that many of these time benefits are due to less rework. Much of the rework, or redesign, in development is caused by changes in specifications and, to the extent that these changes can be avoided, precious time is saved.

Team size is another important factor that distinguishes the fast software developers. The firms reporting the greatest increase in development speed also tend to have smaller teams-- in every stage of the process except one, customer requirements. Given the importance that these firms place on customer requirements, this result is also not surprising.

The insignificance of CASE technologies in our surveys is intriguing and somewhat mystifying. Enormous sums have been poured into the acquisition of tools to support design productivity at every stage of the process-- from systems for configuration management to object-oriented languages and libraries. These tools have clearly had an effect on productivity, but development managers' perceptions are that there are other, more important productivity drivers. We speculate that increasing project complexity and size are obscuring the advantages that CASE tools bring, but our research has been unable to confirm this. If true, then more sophisticated research instruments are needed to prove it.

Given a choice between investing in people or technology, this research suggests that talented people are more important. This echoes the findings of researchers on other types of product development that creative people are the real silver bullet (for example, Curtis, Krasner and Iscoe [11]). The other levers of productivity make a difference at the margin, but they can not overcome a weakness in human capital.

Although much remains to be learned about the process of software development, our results chart a clear path for change. The techniques used by the faster software developers closely resemble the prescriptions proposed for speeding up other forms of product development. Our research suggests that the similarities between software and hardware development may be more important than the differences. The causes, and the cures, for time delays appear to be the same. Continued research to improve the development process should benefit both hard and software.

REFERENCES

[1] M. VAN GENUCHTEN "Why is Software Late? An Empirical Study of Reasons for Delay in Software Development", IEEE Transactions on Software Engineering, Vol. 17 No. 6 pp 582-590, 1991.

[2] G. P. ZACHARY "How Ashton-Tate Lost Its Leadership in PC Software Arena," *The Wall Street Journal*, April 10, 1990.

[3] B. ZEIGLER "New IBM PCs Are Superfast, but Might Be Too Late," *Wall Street Journal*, June 16, 1995.

[4] J. D. BLACKBURN Time-Based Competition: The Next Battle Ground in American Manufacturing, Business One Irwin, Homewood, Illinois, 1991.

[5] G. STALK AND T. HOUT Competing Against Time, Free Press, New York, 1990.

[6] P. SMITH AND D. REINERTSEN Developing Products in Half the Time, Van Nostrand Reinhold, New York, 1991.

[7] J. BLACKBURN, G. SCUDDER, L. VAN WASSENHOVE AND C. HILL "Time-Based Software Development", to appear in International Journal of Integrated Manufacturing Systems, 1995.

[8] J. BLACKBURN, S. BAYLOR, AND G. SCUDDER "Software Development in Japan: Some Preliminary Findings", Vanderbilt University, Owen Graduate School of Management Working Paper# 92-57, September 1992.

[9] G. SCUDDER AND C. HILL "Software Development in Japan: Some Findings from Some Major Companies", Vanderbilt University, Owen Graduate School of Management Working Paper #93-60, August 1993.

[10] J. BLACKBURN, G. HOEDEMAKER, AND L. VAN WASSENHOVE "Interviews with Product Development Managers," Vanderbilt University, Owen Graduate School of Management Working Paper #92-56, 1992.

[11] B. CURTIS, H. KRASNER, AND N. ISCOE "A Field Study of the Software Design Process for Large Systems," Communications of ACM, vol. 31, no. 11, pp. 1268-1287, 1988.

[12] K. MAXWELL, L. VAN WASSENHOVE, and S. DUTTA "Software Development Productivity of European Space, Military and Industrial Applications", INSEAD Technology Management Group Working Paper 94/74, 1994.

[13] F. BROOKS The Mythical Man-Month: Essays on Software Engineering , Addison-Wesley, London, 1975

[14] C. JONES Assessment and Control of Software Risks, Prentice Hall,Englewood Cliffs, NJ, 1994.

[15] L. PUTNAM and W. MYERS Measures for Excellence: Reliable Software on Time, within Budget, Prentice-Hall, Englewood Cliffs, N J, 1992.

[16] S. McCONNELL Code Complete: A Practical Handbook of Software Construction, Microsoft Press, 1993.

Table 1
Spearman Correlation Coefficients--European Sample
Significance level given within (.)
Boldface indicates factors significant at .05 level or better
** indicates significance at 0.10 or better

Project Factors	Correlation with % Reduction in Development Time	Correlation with LOC/CMM Productivity
% of time by stage:		
customer requirements	0.164 (0.13)	**0.303 (0.02)**
planning & specifications	-0.1789 (0.09)**	-0.035 (0.79)
detail design	-0.177 (0.87)	-0.029 (0.82)
coding	0.0013 (0.99)	-0.172 (0.18)
testing/integration	**-0.208 (0.05)**	-0.105 (0.42)
% of effort by stage:		
customer requirements	0.211 (0.09)**	**0.245 (0.05)**
planning & specifications	-0.13 (0.24)	-0.05 (0.69)
detail design	-0.149 (0.283)	0.001 (0.99)
coding	-0.109 (0.329)	-0.18 (0.16)
testing/integration	-0.045 (0.69)	0.006 (0.96)
team size by stage:		
customer requirements	0.0798 (0.476)	0.093 (0.47)
planning & specifications	-0.13 (0.24)	-0.22 (0.09)**
detail design	-0.149 (0.183)	-0.133 (0.30)
coding	-0.11 (0.33)	-0.17 (0.19)
testing/integration	-0.045 (0.69	0.089 (0.492
maximum team size	-0.040 (0.724)	
Project Size (in LOC)	-0.0265 (0.829)	**0.284 (0.025)**
Project Duration	-0.173 (0.10)**	-0.21 (0.10)**
Project Newness	0.078 (0.46)	0.14 (0.27)
LOC/CMM Productivity	**0.366 (0.004)**	
LOC/TMM Productivity	**0.309 (0.016)**	
Project Management Factors		
Prototyping	**0.31 (0.003)**	
Customer Specifications	0.046 (0.664)	
CASE Tools	0.089 (0.40)	
Concurrent Engineering	0.082 (0.44)	
Less Rework	**0.318 (0.002)**	
Project Team Mgt	0.152 (0.152)	
Testing Strategies	-0.20 (0.06)**	
Reuse	0.16 (0.129)	
Module Size and #	0.012 (0.91)	
Communications	0.058 (0.58)	
Better People/Programmers	**0.405 (0.0001)**	

Figure 1
Percentage of Time
in Development Stage

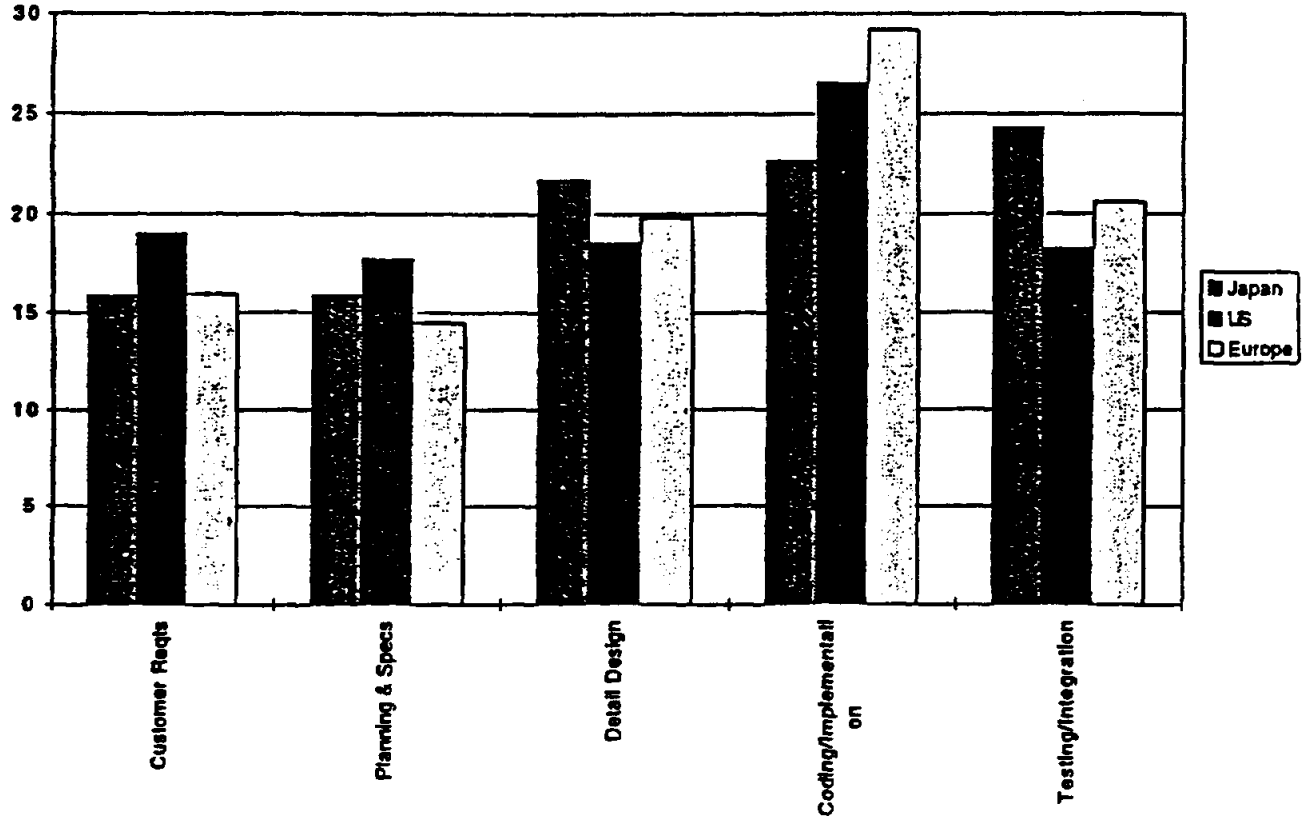
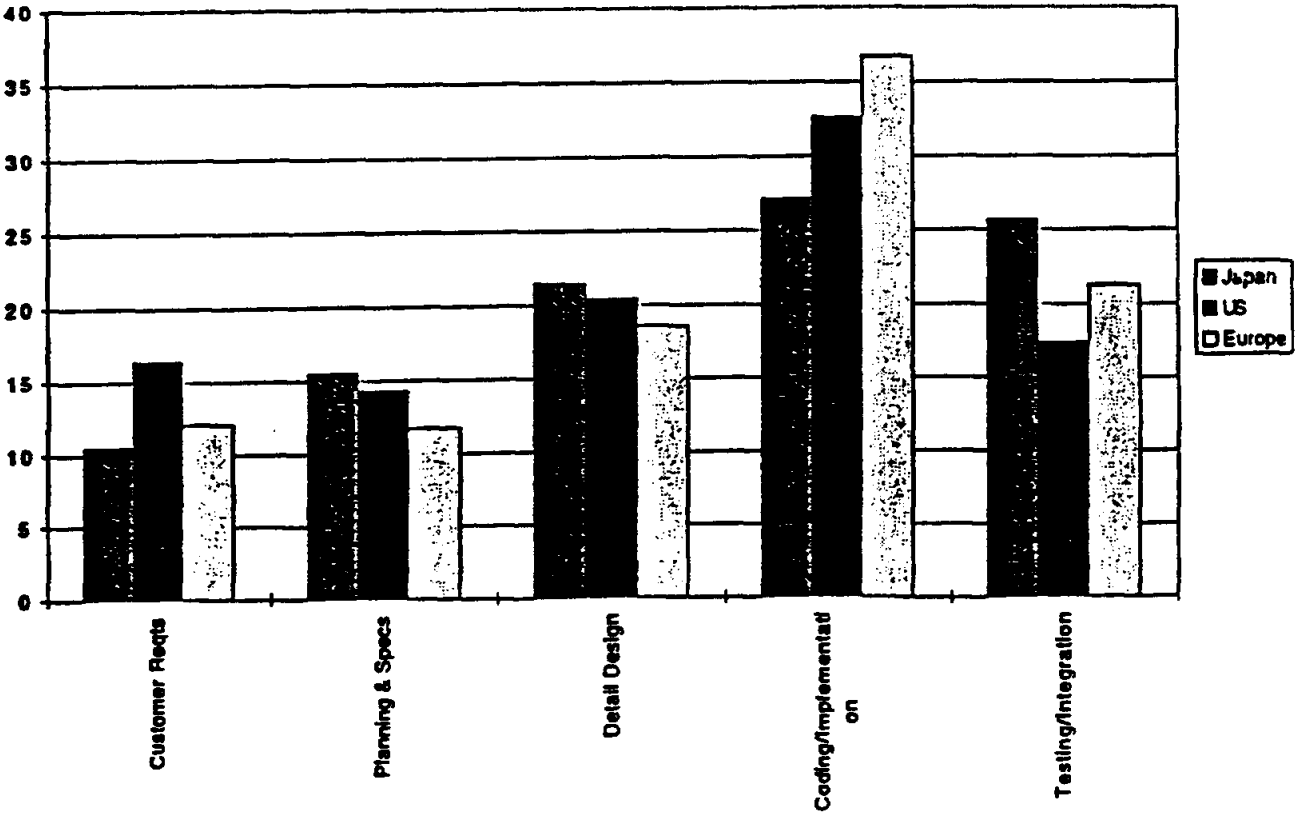


Figure 2
Percentage of Development
Effort by Stage



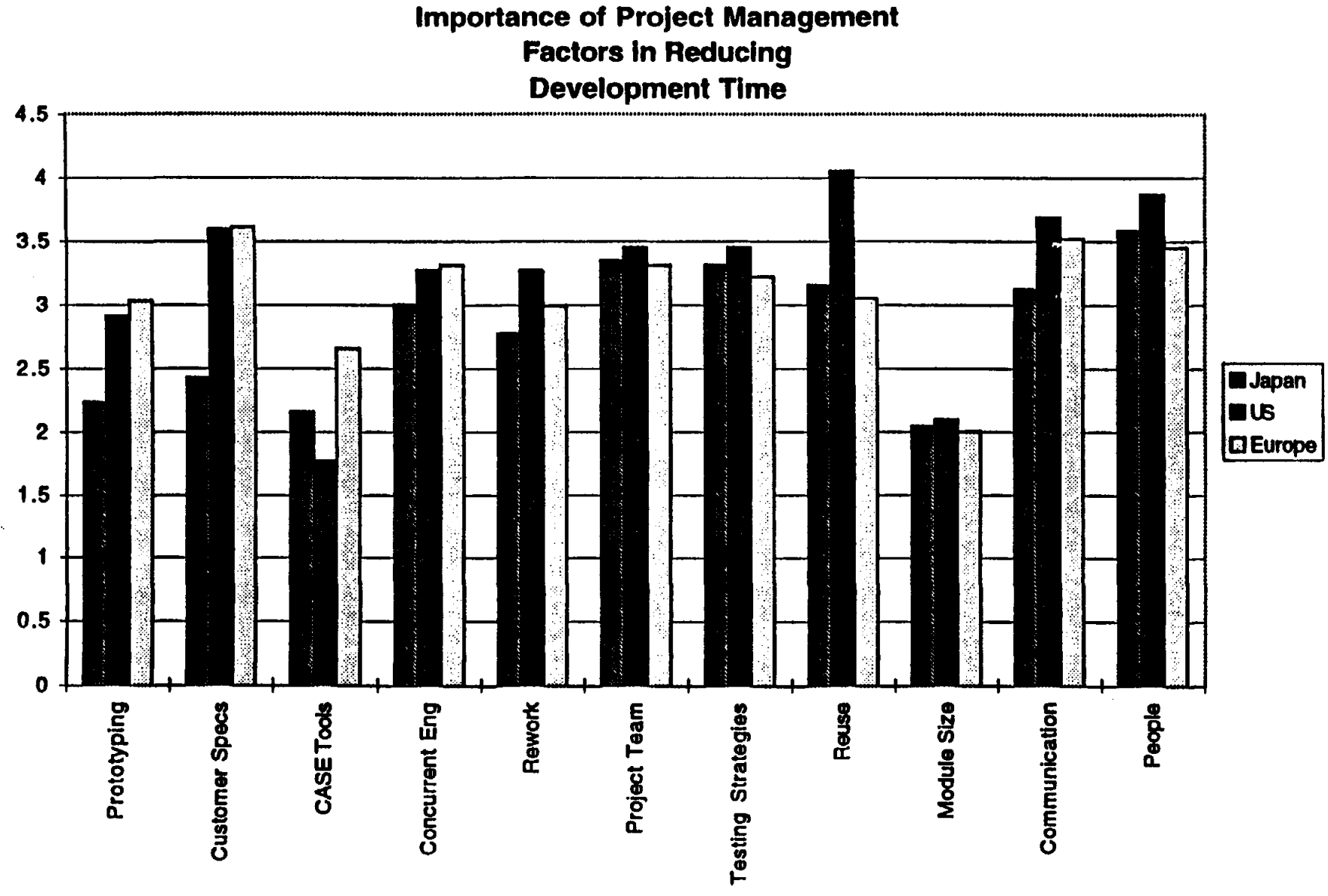

Importance of Project Management
Factors in Reducing
Development Time
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
Prototyping
Customer Specs
CASE Tools
Concurrent Eng
Rework
Project Team
Testing Strategies
Reuse
Module Size
Communication
People
Japan
US
Europe

Figure 4

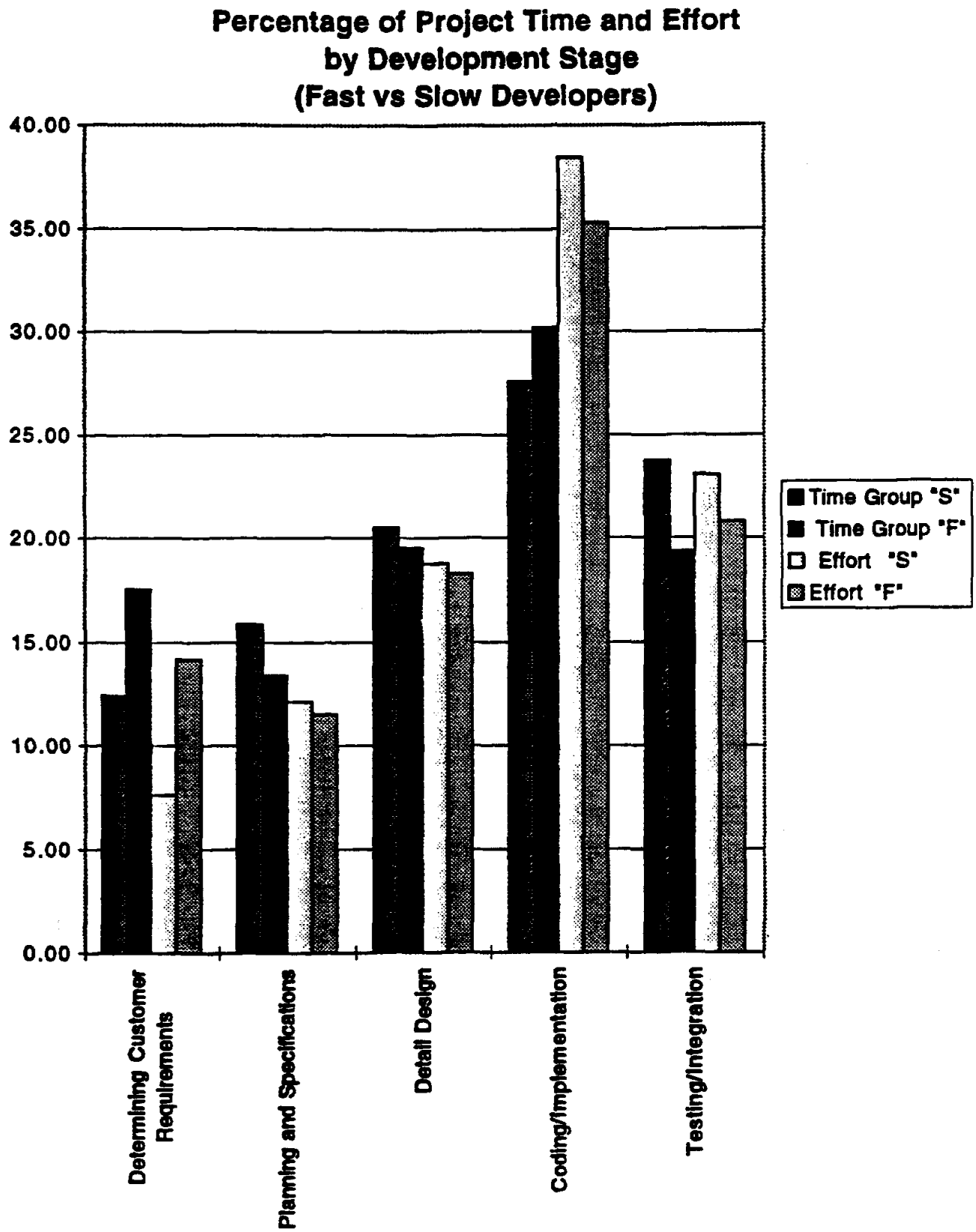

Percentage of Project Time and Effort
by Development Stage
(Fast vs Slow Developers)
40.00
35.00
30.00
25.00
20.00
15.00
10.00
5.00
0.00
Time Group "S"
Time Group "F"
Effort "S"
Effort "F"
Determining Customer Requirements
Planning and Specifications
Detail Design
Coding/Implementation
Testing/Integration

Figure 5

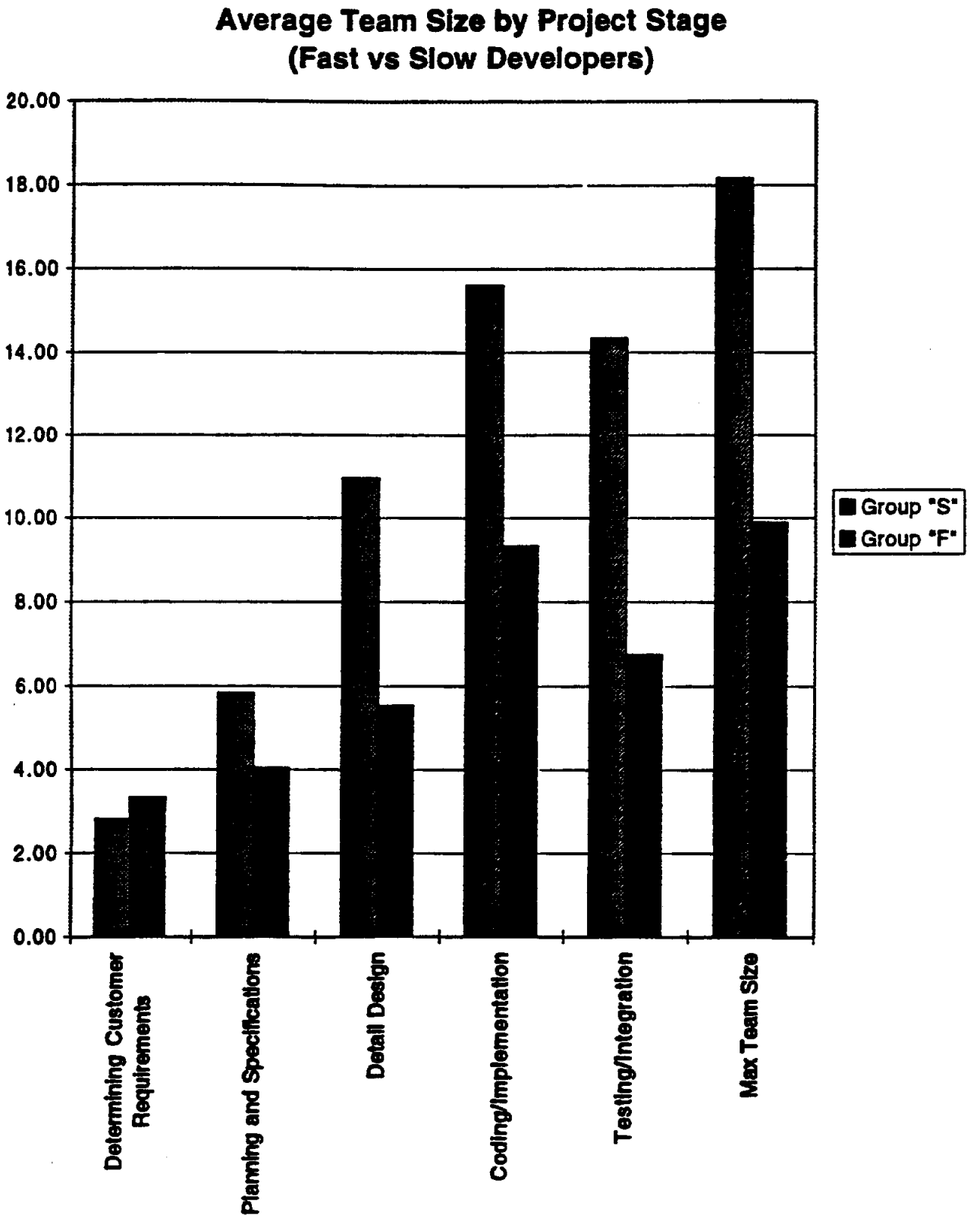

Average Team Size by Project Stage
(Fast vs Slow Developers)
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
Group "S"
Group "F"
Determining Customer Requirements
Planning and Specifications
Detail Design
Coding/Implementation
Testing/Integration
Max Team Size

Figure 6

Average Productivity of Fast vs Slow Developers