Independent Research Study of Software Re-Use

(Using Frame Technology)

September 1994

QSM Associates, Inc.
8 Meadow Ridge
Pittsfield, MA 01201
Phone: 413-499-0988
Fax: 413-447-7322

Table of Contents

Abstract

Proponents of software re-use, object oriented methods, and languages that implement object features (i.e. Ada), make a strong, and largely common sense argument that, to build software systems, it costs more, takes longer, and is more error prone to always build these systems from scratch when it is unnecessary. In short, “re-inventing the wheel” consumes valuable time and effort, and doesn’t necessarily ensure that you’ll have a better wheel in the end.

And in software development, as in many other disciplines that require engineering design processes, statistics show a nonlinear, escalating, relationship between the cost, schedule, and expected defects as a function of increasing size. Therefore, the reverse also holds true; namely, that anything you can do to decrease the amount of “building from scratch” will yield substantial reductions in these areas.

Hence we have several modern initiatives that emphasize frameworks for building “library assets”, be they things like object libraries, frame libraries, and the like, which can be called upon to furnish common components when needed. Not that the concept is new (e.g. subroutine libraries); indeed, most programmers and engineers likely have some kind of re-use as part of their skill sets, sometimes being as fundamental as personal work files or subroutines that they utilize every day.

But what <u>is</u> new in modern times is the *automation* of re-use philosophies, to bring such libraries out of individual desks and encourage the building of departmental, divisional, or even corporate assets. This multiplies the potential for re-use. At the first level, it opens potential for sharing of assets across an enterprise, not simply applying re-use on a singular, individual level.

On a second, and more powerful level, re-use takes on another dimension when implemented as part of a systematic strategy through the building of a re-use infrastructure. Most organizations are not even near touching this potential. Indeed, in the early days of the SEI Maturity Framework, re-use was initially considered a discipline that was characterized at the highest maturity level, SEI Level 5 (this has since been re-thought however). Yet, some are implementing software re-use through various technologies that employ such methods. Frame technology, by its definition, is one of these methods that encourages the building of re-usable sub-assemblies, or frames. (For specific details on this subject, refer to the bibliography of this report.)

This assessment sought to identify what some of the real world gains are for software re-use, through the analysis of actual project data, as opposed to extrapolations on a hypothesis. As an independent metrics organization, QSM was asked to provide an unbiased metrics assessment of software project behavior, for a diverse group of companies all building applications via re-use using frame technology. The report you now hold is the result of this analysis, conducted during July/August 1994.

Introduction

This research study is an independent assessment of the software development cost, schedule, and quality benefits of implementing systematic software re-use for application development. In particular, QSM analyzed projects built in the real world; 15 separate projects collected independently from 9 different organizations. Re-use was accomplished in these organizations through application of frame technology as implemented in the NETRON/CAP tool from Netron Inc., Toronto, Canada. The specific objective was to benchmark the bottom line productivity effects across the 9 organizations, and compare them against industry. We wanted to identify what patterns, if any, were present; both individually by company, and for the overall group as a whole.

To meet these objectives, the 9 organizations that elected to participate in this research study each contributed project data, using an identical set of guidelines that were developed and refined by QSM research over the last 15 years. These guidelines are embodied in QSM's industry data collection framework, and comprise the "front end" of the QSM Productivity Analysis Database System (PADS®). When complete, the data was sent directly to QSM Associates, Inc. (herein referred to as QSM) for validation and analysis. None of the information gathered by QSM was disclosed in its raw form to Netron or any other party outside of QSM. Two of the 9 contributing organizations requested that their identity remain anonymous. The other 7 organizations which participated in and funded this study are:

Ameritech
Chemical Bank
Hudson's Bay Company
Noma - Cable Tech
Revenue Canada
Teleglobe Insurance Systems
Union Gas Limited

A special note of appreciation is due to each of the organizations that contributed their valuable time and energy to furnishing the information for this analysis. The results of this effort are made possible by their active participation.

To evaluate process productivity, the QSM Productivity Analysis Database System (PADS®), was used. PADS is an international database of more than 3,800 completed software projects collected worldwide (6/1/94 Baseline), and contains validated productivity statistics from hundreds of leading software producers throughout the industry from the U.S./North America, Europe, Japan, and Australia. PADS provided benchmark statistics against which the contributing organization's projects were compared.

This report is divided into 2 main sections. Section 1 (contained herein) provides analysis findings on the data sample in an aggregate format, with no connections made between project specifics and the contributing organizations (this portion of the report will be public domain). This "aggregate view" of the analysis was done to honor the confidentiality of the specific data provided by the participants. Section 2 contains an individual, private assessment for each contributing organization. These confidential appendices highlight findings that are specific to projects of that organization. This portion is considered proprietary and is for the sole, express use of the contributing organization. As such, each company unique appendix has only been provided to the organization that furnished the data.

All of the contributing organizations provided data for projects which were components of their strategic computer applications for business operations. Therefore, the project data was compared against the Business/MIS Systems application segment of the QSM database.

The data analyzed for this report was evaluated by Mr. Ira Grossman and Mr. Michael Mah of QSM Associates, Inc. As of this writing (September 1994) it is the first known independent metrics assessment of the cost, schedule, and quality benefits of systematic software re-use for application development. For those who would like to direct any inquiries regarding the study or the methods that were employed, kindly contact our offices at:

QSM Associates, Inc.
8 Meadow Ridge
Pittsfield, MA USA 01201

Tel (413) 499-0988
Fax (413) 447-7322
e-mail: Ira Grossman, 72714.3006@compuserve.com, or
Michael C. Mah, 74051.2111@compuserve.com

Executive Summary

Based on analysis of the 15 projects in this data sample, the following conclusions can be drawn:

- ❑ An average size project in this sample of 133,000 lines of code took approximately 5.3 months to build, versus an industry norm of 18 months. This represents a 70% cycle time reduction.
- ❑ All 15 projects consistently positioned as high performers against QSM Industry Trendlines for schedule. On average, they positioned in the top 84th percentile.
- ❑ The same average size project typically expended 27.9 person-months of effort, versus an industry norm of 179.5. This represents an 84% reduction in cost, amounting to a savings of approximately $1,389,000 US.
- ❑ All but 2 of the larger projects positioned in the top 84th percentile for effort (low cost performers) against the QSM Industry Trendlines. The remaining 2 applied large teams in an attempt to further compress schedule, and expended more effort.
- ❑ Six of 7 projects that provided defect statistics nominally experienced average to lower than average defects found and fixed during testing.
- ❑ All but 2 projects utilized far fewer staff than industry average, applying 10 people or less per project.
- ❑ The QSM Productivity Index (PI), a metric of project efficiency and complexity, was derived for each of the projects. The average PI for the sample was 26.2. This average is much higher than the industry average PI of 16.9 from QSM's database statistics; an average gain of 9.3 PIs. 47% of the projects sampled position in the 95th percentile for this metric.
- ❑ This high productivity is directly correlated to the high volume of functionality generated through re-use via frame technology.
- ❑ All but 2 projects produced high degrees of functionality per person-month, and all produced high degrees of functionality (and code) per month. Two projects expended higher effort due to schedule pressure (large teams), resulting in lower functionality per person-month.
- ❑ PIs tended to increase on a company basis as their experience with the re-use technology improved. First time users experienced an average PI of 25.6, intermediate users 26.0, and mature users 27.4.
- ❑ Smaller projects (<100,000 lines of code) tend to spend less time and less effort in the Functional Design phase relative to the industry average. Larger projects (>100,000 lines of code) were closer to the average.
- ❑ Project overruns for this sample were nominal. However, nearly one third (27%) of the projects were under budget and ahead of schedule. This contrasts with only 2% of industry exhibiting project underruns.
- ❑ Both staff turnover and requirements change were low, less than 10% and less than 13% respectively. Overall team skills and experience were also medium/high. These were favorable factors which on a secondary level contributed to the high QSM PIs that were achieved.

SECTION 1 - AGGREGATE PROJECT BEHAVIOR

Measurement Techniques and Projects Assessed

Method of Evaluation

The project data were evaluated by QSM using the general method shown in Figure 1. This involved:

❑ Providing each contributing organization with data collection forms.

❑ Telephone interviews with project managers for follow-up discussions to verify data accuracy and completeness.

❑ Extensive analysis of the data to position the projects against industry reference measures.

❑ Preparation and delivery of this report.

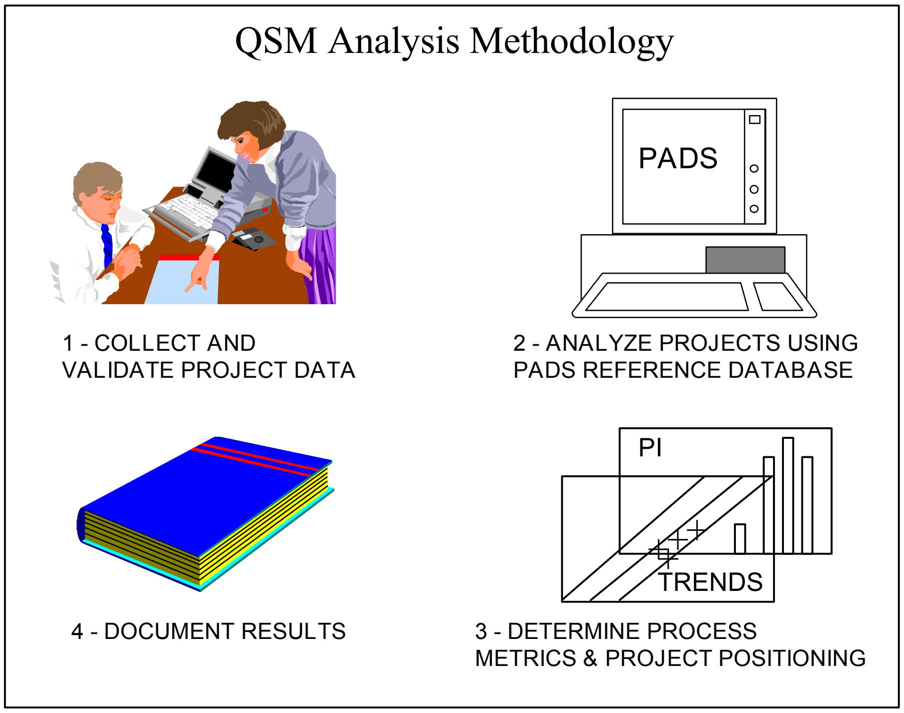
QSM Analysis Methodology
PADS
1 - COLLECT AND VALIDATE PROJECT DATA
2 - ANALYZE PROJECTS USING PADS REFERENCE DATABASE
PI
TRENDS
4 - DOCUMENT RESULTS
3 - DETERMINE PROCESS METRICS & PROJECT POSITIONING

Figure 1. QSM Analysis Methodology.

Data Collection

A data collection form was provided to each contributing organization. This form lists all of the data capture fields contained in PADS. The data includes both quantitative measures (hard factors) and qualitative measures (soft factors). The quantitative measures include system size (both lines of code and function points were applicable), schedule by project phase, effort by project phase, and error statistics. The qualitative measures focus on experience, environment, and other project specific factors which influenced the software behavior. (A copy of the data collection form is available by contacting QSM.)

The intent of the data collection forms is to obtain information to derive measures for process efficiency (productivity), while also profiling the environmental attributes unique to each organization. The data was collected over a 2 month period to accommodate the schedules of all the participating organizations. The average time spent on each project's data form by contributing organizations was in the range of 4 to 8 hours.

Data Validation

QSM collected project data on 17 projects developed using NETRON/CAP. Two of the 17 projects were omitted from Section 1 of this study. One of the projects was a very small prototype effort (only 1 to 2 weeks), and the other was a project which was incomplete, without an estimate of how much functionality would be delivered at completion. The remaining 15 projects included in this study were rigorously evaluated over a 1 month period (overlapping with data collection), to ensure data completeness and accuracy.

For project size metrics, both function points and lines of code (new and modified, excluding comments and blanks), were used. Only 4 of the 15 projects (3 organizations) included function point data. The others did not have counting methods in place for this metric. All 9 organizations did provide line of code counts in accordance with counting rules which are consistent with the SEI's recommended approach for counting logical source statements.

To simplify and ensure consistency and accuracy in the counting methods for lines of code, a utility was provided by Netron which scans the source files , and counts both the NETRON/CAP Specification or (SPC) code written (which goes into the "front end" of the tool), and the final assembled Cobol source (produced out of the "back end " of the tool). It also provides counts on comments and blanks so that they can be removed from the source code counts.

The approach taken in this study to calculate metrics of process productivity, will use the final assembled code as a measure of project size and functionality. A question was raised regarding possible "excess code" produced by the tool which might artificially inflate the results. We queried both Netron and study participants about this, and found consensus that there was likely no more than 10% "overhead" or unexecutable code associated with the frame libraries. This is similar to what naturally occurs in traditional hand programming. It does not have any significant impact on the results presented in this study. In fact, even if we conservatively reduced the generated code size by 30%, the resultant process productivity calculations are still very high.

In Section 1 of this report, Industry Trend Line graphs that depict project behavior will be shown with Effective Source Lines of Code (ESLOC) as the X-axis of each graph. The management metric under scrutiny (schedule, cost, quality) is depicted on the Y-axis. (For those organizations contributing Function Point data, Section 2 shows selected Industry Trends applicable to each organization using Function Points as the size measure.) It is important to note that the findings contained in Section 1 of this report are conclusive using either sizing metric.

Projects Assessed

The 15 NETRON/CAP projects assessed were contributed by 9 distinct organizations across the U.S. and Canada. The projects were comprised of many different I/S applications including customer tracking, sales and financial analysis, data interchange, retail, training, and general business support functions. All of these projects were categorized as Business Systems/MIS for comparison against QSM industry benchmarks.

The 15 projects totaled 12,610,948 effective lines of delivered code (ESLOC = new plus modified/generated lines of code). The 4 projects contributing function point data totaled 18,942 function points. These 4 projects had between 10 and 44 lines of NETRON SPC (handwritten code) per function point. This is consistent with the ratios indicated by other researchers in the industry for “Program Generators”. However, the ratio of delivered lines of code to function points was significantly higher, from 100 to 700 lines of code per function point. This is indicative of the variation in possible frame types, and the significant functionality provided by the re-use of frames.

The projects ranged in size from under 5,000 lines of code to almost 10,000,000 lines of code, with an average size of 840,730 lines of code (The “average size” used to compute the schedule and cost savings in the Executive Summary omitted the 2 largest projects in the sample. This is explained in more detail in the section on Schedule Benefit and Cost Savings). A total of 2,229 person-months of development effort were expended on these projects which were all completed between 1988 and 1994.

Table 1 shows a summary of the projects and their respective size, time, effort for the Main Build phase (detailed design, construction, and test), and the calculated Process Productivity Index (PI - described on the following pages and in Appendix A.)

PROJECT NAME	TIME (MOS)	EFFORT (STAFF-MOS)	SIZE (ESLOC)	Productivity Index
PROJECT 1	1.5	2.1	4574	19.8
PROJECT 2	4	16	306480	30.3
PROJECT 3	1	1.5	15484	27.6
PROJECT 4	1.5	1.5	31313	29.1
PROJECT 5	2.5	2	23134	24.2
PROJECT 6	4	16.5	127174	26.6
PROJECT 7	25.5	1737	9300715	27.7
PROJECT 8	4	4	52611	24.8
PROJECT 9	.75	.75	8475	27.6
PROJECT 10	1.1	1.1	24926	29.9
PROJECT 11	10	260	476710	23.2
PROJECT 12	6	6	192161	27.5
PROJECT 13	15	147	1576891	26.7
PROJECT 14	13	20	421000	24.8
PROJECT 15	4.4	13.1	49300	22.4

Table 1. Main Build Data for Projects Assessed.

Quantitative Assessment

Macro Management Measures

The data in Table 1 was part of an extensive data collection effort comprised of schedule and effort data related to the projects, along with qualitative information related to the environment, skills, and general project characteristics. After data validation and verification, this information was used as input to PADS. For each project, PADS then calculated an important management indicator, the QSM Productivity Index (PI).

Productivity Index

The Productivity Index is a measure of process productivity and development complexity. It is derived from quantitative project data and, in its interpretation, can encompass numerous factors affecting the software development environment, including management strategy, development methods and technical aspects. For details on the derivation of the Productivity Index see Appendix A and the Bibliography of this report.

The Productivity Index goes far beyond older and perhaps more traditional measures for productivity such as function points per staff month, or lines of code per staff month, which by their nature do not have any schedule component, since they are simply "output per unit of effort", irrespective of time.

The average PI for Business applications in the QSM industry database is 16.9, based on data through mid-1994. Projects with PIs below 16.9 are either more complex and/or have a less efficient development environment; those above 16.9 were less complex applications or were developed in a more efficient environment than the average project.

An increase or decrease of one value on the PI scale has a measurable impact on time as well as effort. Table 2 illustrates the magnitude of a PI shift on a typical Business/MIS software project made up of 85,000 new and modified lines of code. For these examples, the calculations apply only to the Main Build phase of the software development.

PI	Minimum Schedule	Effort (PM)	Cost (x 1,000)	Peak Staff
14	16.0	164	$1,300	16
15	14.5	125	$1,000	13
16	13.0	88	$ 730	10
17	12.0	67	$ 560	8

Table 2. Impact of PI on Time and Effort.

Some Rules of Thumb: In a typical Business/MIS software project, compared to the industry average:

- ❑ An organization one PI below the average takes 10% more time and costs approximately 27% more.
- ❑ An organization one PI above the average takes 10% less time and costs approximately 27% less.

Figures 2 and 3 show the PIs calculated for the 15 projects assessed as well as how they compare against industry. The data are shown in frequency distributions to highlight their central tendency.

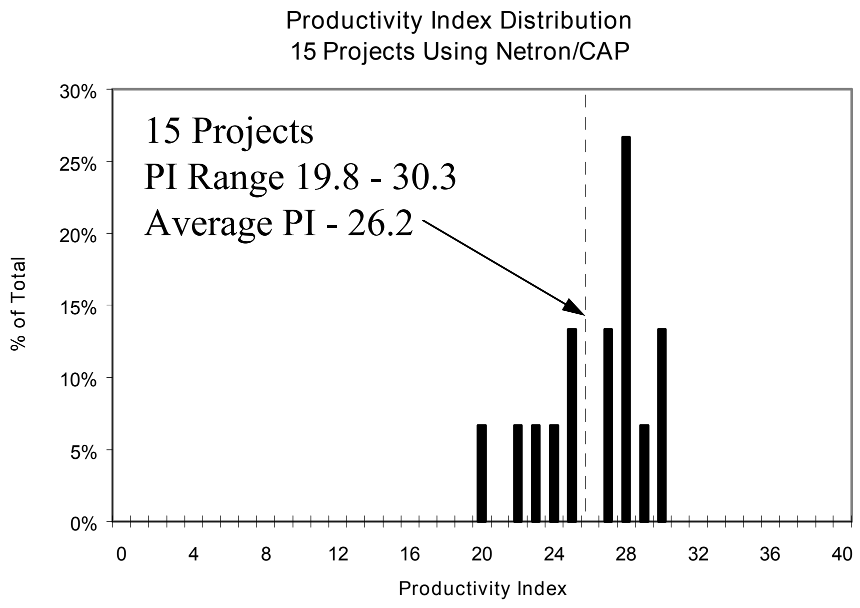

Figure 2. Productivity Index Distribution for 15 NETRON/CAP Projects.

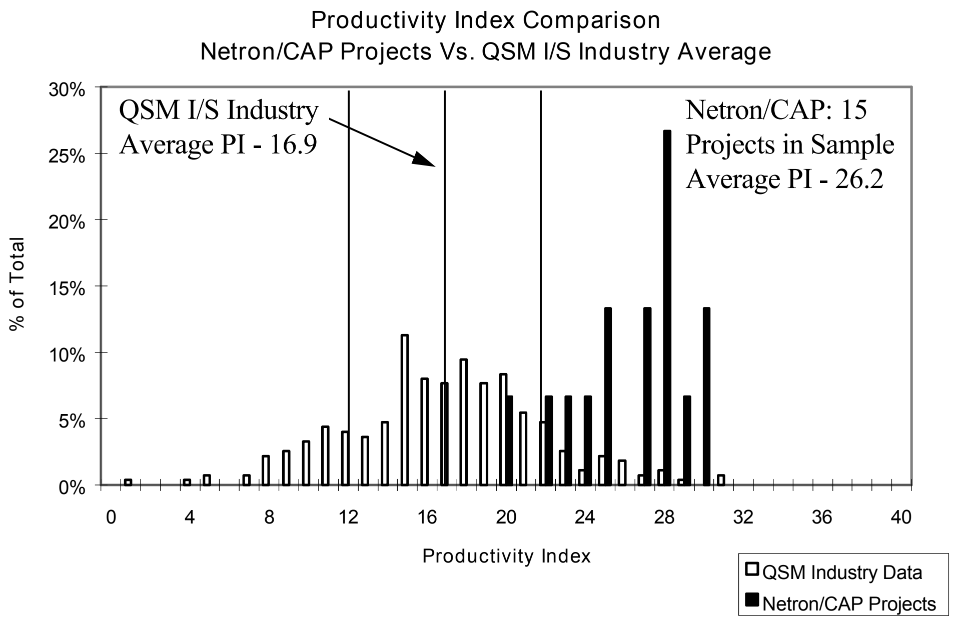

Figure 3. NETRON/CAP Project PIs vs. Industry Data.

Productivity Index Observations

Overall PI Distribution

The range of PIs depicted in Figure 2 is from 19.8 to 30.3 with an average value of 26.2. This compares very favorably against the industry average PI for I/S systems of 16.9. The I/S industry average is comprised of the DP/MIS subset of the overall QSM data base which contains the most contemporary (1993/1994) projects developed by Fortune 500 class companies using traditional hand coded methods, 4GLs, Case Tools, and Application Generators.

In Figure 3, the shorter hollow bars represent the distribution of PIs for QSM's industry data. The center line on the graph represents the industry average PI, and the lines above and below the average represent plus and minus one standard deviation from the average (68% of all historic I/S project PIs fall within this range).

100% of the NETRON/CAP project PIs position above the industry average. All but 1 project (93%) position above the +1 Standard Deviation (or sigma(σ)). This puts the NETRON/CAP projects well into the 84th percentile (or top 16%) of the database as a group. Even more favorably, a high 47% are in the upper 95th percentile (at or above $+2\sigma$). This means that almost half of the projects in this sample have PIs that have only been exhibited by 5% of the best organizations we've examined in 16 years of collecting I/S data worldwide.

Average PIs by Organization

Figure 4 shows the individual or average PI (for organizations contributing more than 1 project, the average PI was calculated) for the 15 NETRON/CAP Projects. The average PIs are contrasted against both the sample average for the 9 companies, and the overall industry average. All company averages are well above the overall industry average, and all but 1 project in the sample (from Company C - an older project at a PI of 22.4) are within 1.5 PIs of the sample average. These PIs by company are very tightly distributed considering the diversity of the participants, representing the telecom, banking, retail, government, insurance, utilities, and manufacturing industries. This suggests that NETRON/CAP is adaptable across a rather wide spectrum. The companies exhibited consistently high process productivity independent of the development organization or application specifics. Using the PI as a quantification of this increased process productivity, we can translate to bottom line management terms of schedule, cost, and quality.

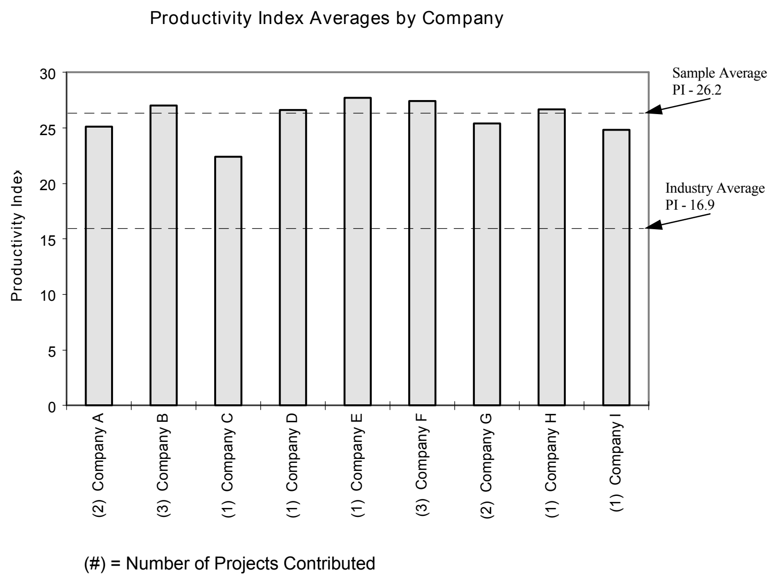

Figure 4. Productivity Index Averages by Company.

Schedule Benefit and Cost Savings

This sample of projects has an average PI that is 9.3 values above the overall I/S industry average (based on the most contemporary sample of I/S projects collected by QSM through mid-1994). The cost and schedule implications of this PI advantage are significant. To quantify this, we've taken several steps: 1) Calculate the average size of the projects in the NETRON/CAP sample (the 2 largest projects were omitted from the average size calculation because they were an order of magnitude larger than the other 13 projects. This results in a conservative benefit calculation). 2) Calculate the average team sizes for both Functional Design (software requirements and high level design) and Main Software Build (detailed design through code, integration, test, and ready for production). 3) Use QSM's Software Lifecycle Management model (SLIM®) to emulate the cost and schedule for a project of this average size, first at a PI of 16.9 (industry average), then at a PI of 26.2 (NETRON/CAP sample average). 4) Compute the difference.

Because each integer rise in the PI has a 10% schedule impact on the previous value, and a 27% cost impact, the savings realized in comparing a PI of 16.9 to 26.2 are dramatic. Compared to traditional Cobol systems development, building an application at a "sample average" size of 133,000 lines of code (with a peak staff of 5.5 people) using frame technology via NETRON/CAP saves about 12.7 months, a 70% schedule reduction. (Note: The sample average size is the average for the majority of projects in the study, omitting the 2 largest systems (1.5M and 9.3M lines of code). As such, these savings are conservative, although the percentages remain the same.)

It must be noted that to achieve this level of cycle time reduction by brute staffing alone is not only unfeasible (it would require in excess of 125 staff), but it is also statistically impossible, based on the latest industry data.

The effort expenditure is reduced by approximately 152 person-months, down from 179.5 person-months to 27.9 person-months (an 84% cost reduction). In pure dollars, this translates to $1,389,000 US using a fully burdened labor rate of just over $9,000 per person-month.

In addition, it would take 12.7 months longer to reach the same level of application quality that is realized in 5 months on the NETRON/CAP projects.

These schedule, cost, and quality figures represent the average savings per system for the overall sample. The average savings per system for each individual organization will also be calculated in Section 2 of this report.

Figures 5 and 6 show the schedule and cost at the industry average PI of 16.9 versus the sample average PI of 26.2.

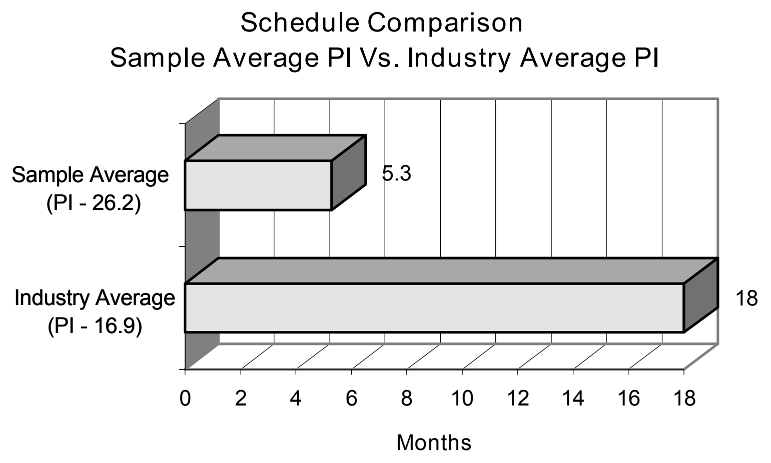

Figure 5. Comparing Sample Average Schedule Vs. Industry Average Schedule.

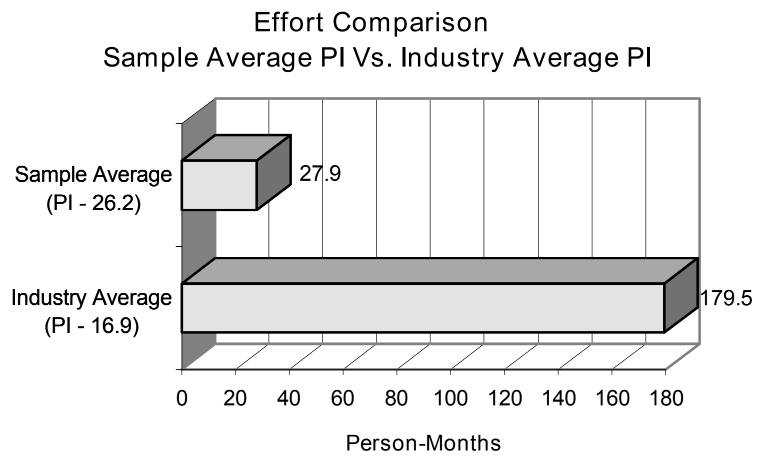

Figure 6. Comparing Sample Average Effort Vs. Industry Average Effort.

PI as a Function of User Maturity

The sample of 15 projects in 9 organizations was divided into 3 classifications of "Netron User Maturity". Those organizations using NETRON/CAP on an initial project or pilot were categorized as First Time Users. Five organizations, each contributing 1 project, were included in this category. The Intermediate Users category was typical of companies using Netron on several projects, but still integrating the approach into their overall software development standards. This category contained 3 companies and 7 projects. Lastly, 1 organization was categorized as a Mature User. This organization has been using NETRON/CAP for several years and has well established frame libraries. This organization contributed 3 projects.

As seen in Figure 7, the average PIs calculated in each of these 3 categories shows an improvement trend from First Time User to Mature User. However, 60% of the First Time Users had PIs above the sample average of 26.2, and the lowest PI in this group was still 5.5 PIs above the industry average. This indicates that it does not take years of investment to achieve high process productivity with software re-use.

It should be noted that the Mature User category had the smallest sample size, in that it represents just 1 organization. In addition, this organization reported high skills and experience in all categories surveyed (tools, methods, utilities, languages, application, etc.), which may have also contributed to the higher average PI independent of NETRON/CAP. Further analysis with additional data is required to assess productivity levels that can be reached for organizations like this.

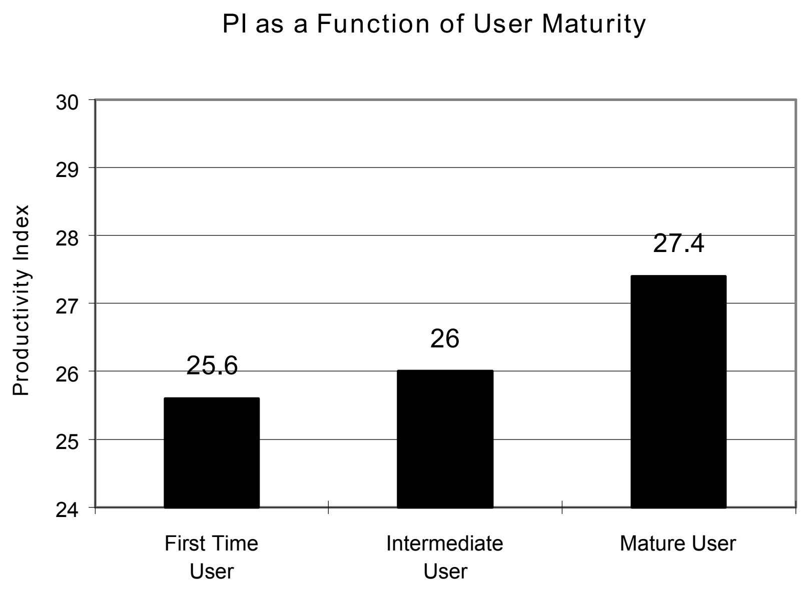

Figure 7. Productivity Index Averages by User Category.

Industry Trend Lines

Composition

QSM Industry Trend Lines have been derived from the more than 3,800 systems in the QSM database, and can be used to assess management strategy, development productivity and product quality. This is done by comparing data from a particular company against others in their industry for individual performance metrics.

This diverse body of industry trends are constructed using reference metrics within a specific application category (e.g. Business, Real Time, Telecommunications, etc.). Standard slopes for the trend lines have been determined from the data. The trends include:

- ❑ Resource and Schedule Metrics
- ❑ Reliability Metrics
- ❑ Size Metrics
- ❑ Design Phase Metrics
- ❑ Analysis Phase Metrics

The trend lines have been verified by examining data from independent organizations worldwide for more than 15 years. They have been refined many times, are reliable, and well-represent software behavior within the industry's latest level of recording and measurement accuracy.

Each of the trend line graphs contains three lines. The middle line is the average behavior for the software metric as the size increases. The top and bottom lines are the plus and minus 1σ bounds. This means that 68% of the historic projects in the QSM database reside between the upper and lower lines.

The trend lines use logarithmic scales. This is for efficient plotting of all the data on one graph. More importantly, it also results in the fact that small distances from the average trend correlate to large differences in absolute values for the metric being assessed (schedule, cost, quality, etc.). Appendix B includes the trend line reports, so that the actual data values for the project sample and the industry average may be reviewed.

NETRON/CAP Project Trends

The NETRON/CAP project data are positioned against the Business Systems industry trends. Trends are presented on the following pages for all 5 categories described above. Appendix B contains copies of all the PADS tool's graphs and reports.

Main Build Resource Metric: SCHEDULE

This graph shows the Main Build phase (detailed design, construction, test and validation) for schedule in months. The horizontal axis represents lines of code (new and modified); the vertical is schedule (elapsed calendar months). If your projects position below the average line, they are being built faster than the industry norm for this class of work.

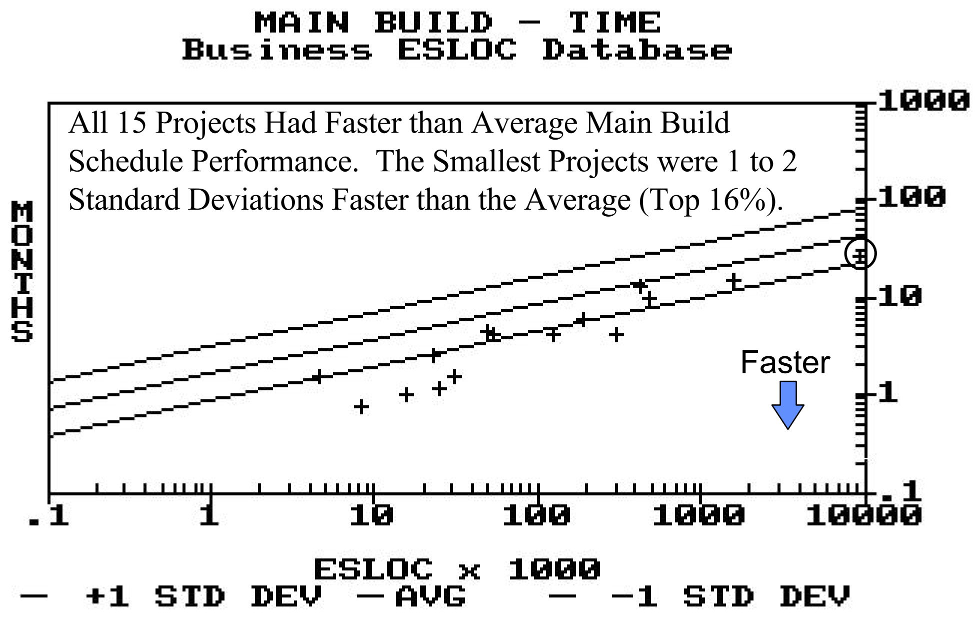

Figure 8. Main Build Schedule Trend.

The Main Build phase is traditionally where the largest percentage of development time and effort is spent. In this sample of 15 projects, 63% of development time and 90% of development effort was expended in this phase. When positioning the 15 projects developed using NETRON/CAP against this Main Build schedule trend, it is clear that these projects all achieved rather remarkable schedules, with several much faster than the industry average.

Furthermore, across a very wide size range, from under 5,000 lines of code to almost 10 million, in 9 different organizations with varied applications and environments, the schedules are consistently faster than the average.

Four of the smallest projects in the sample, at an average size of 20,000 lines of code, with teams of only 1 or 2 people, had schedules which were 2 standard deviations faster than the average (the top 5% of the database). Two of these 4 projects were enhancement and conversion efforts which tend to be less complex than new developments. These project had a high average PI of 28.6.

All of the larger projects on the right side of the trend (from 100,000 lines of code on up) also had competitive schedule performance. The project closest to the average used a very small team of people (peak of only 4, versus an industry average closer to 15) which tends to take longer. However, as seen on the cost trend and the quality trends, this strategy yielded excellent results in both of these domains (see Figures 9, 11, and 12).

Main Build Resource Metric: EFFORT

This graph shows the main build effort in person-months. The horizontal axis represents lines of code (new and modified); the vertical is effort (measured in person-months). If your projects position below the average line, they are expending less effort and, costing less than average to build.

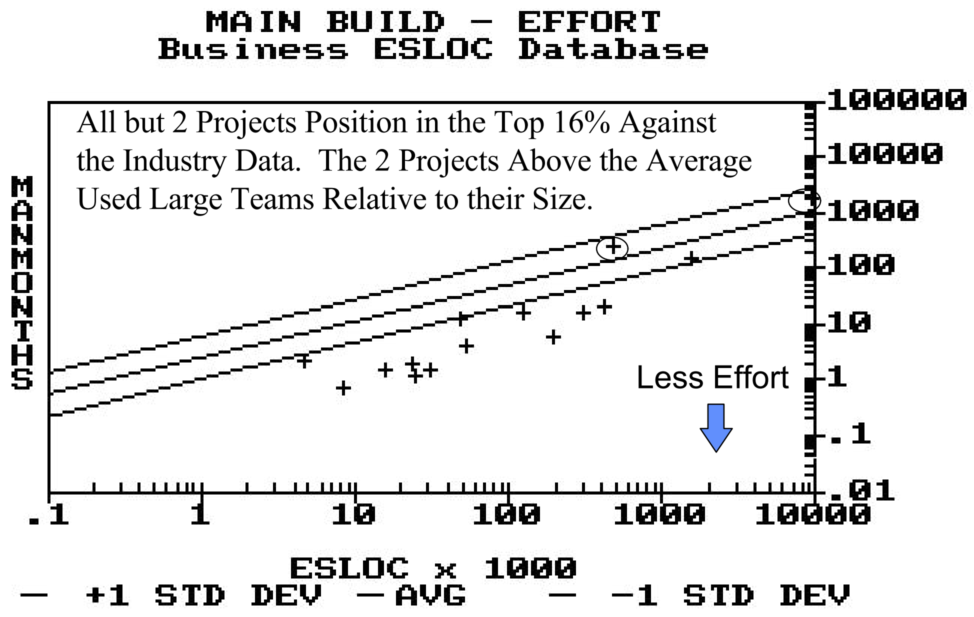

Figure 9. Main Build Effort Trend.

This trend shows all but 2 projects in the NETRON/CAP sample expended 1σ to 3σ less effort than the average. In terms of cost-effectiveness, these 13 projects position at the top 16% of the database. Nine of these 13 are in the top 5%. This is significant from a purely economical standpoint.

Two of the larger projects in the sample used larger than average team sizes (see Figure 10 - Average Manpower Trend). The larger teams resulted in good schedule performance as seen on the previous schedule trend, but resulted in effort expenditures 1/2 standard deviation above the average. A large team strategy is typical of projects with schedule pressure, and these projects were indeed 2 of only 3 in the total sample that experienced a schedule overrun. While the actual schedules achieved on these projects were credible, the staffing strategy used was more costly. We also would expect that the haste resulted in higher error rates (although no error data was furnished for these 2 projects that would enable us to confirm these quality impacts).

The 8 smaller projects (under 60,000 lines of code) had an average effort expenditure of only 3.3 person-months. The 7 larger projects (over 100,000 lines of code) had an average effort expenditure of 315 person-months. The overall average for the sample was 149 person-months.

Main Build Resource Metric: AVERAGE STAFFING

This graph shows the main build average staffing (person-months/month). The horizontal axis represents lines of code (new and modified); the vertical represents the average staff level. If your projects position above the average line, they are using larger teams than the average.

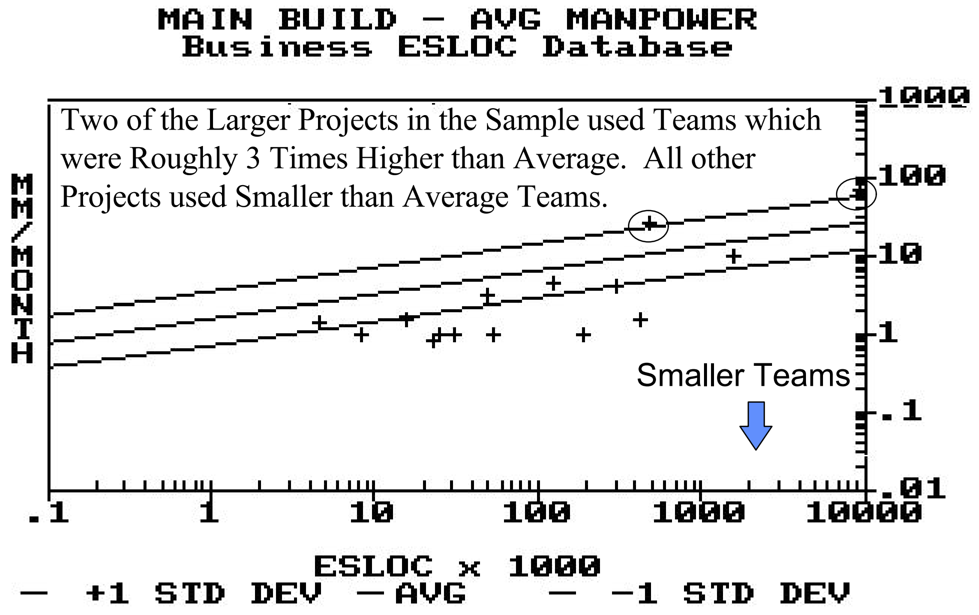

Figure 10. Main Build Average Staff Trend.

This trend is directly linked to the previous effort trend and shows the average staffing in the Main Build to be below the average in all but 2 instances. Overall, the companies in the study are using smaller than average teams while leveraging NETRON/CAP to develop projects of varied sizes. In spite of the smaller team sizes, they're still achieving very short schedules. This also results in the very low effort numbers as seen on the previous trend.

As was explained on the effort trend, 2 of the larger projects used larger than average teams relative to the application size. To highlight this, we'll compare the third largest project, at just under 500,000 lines of code (Project X), to the next largest project, at just over 1.5 million lines of code (Project Y).

Project X used a peak staff of 40 people, (with an average staff of 26 people) during the Main Build phase. In contrast, Project Y was 3 times larger in code size, but used a smaller peak staff of only 33 people (with an average of 10 people during the Main Build). While these projects were both 30% faster than industry average at their respective sizes (see Figure 8), Project X used more of a "brute force approach" to get the schedule via aggressive staffing. While this resulted in a schedule compression of about 2 calendar months, it cost money in both dollars and labor, by about a factor of 2.

Interestingly, the organization that developed Project X also provided data on a second project which was smaller and considerably less complex. In this instance, the organization took a different staffing approach, using, a much smaller team. It therefore appears that the "brute force" staffing style exhibited on Project X may have been project specific to the schedule pressure that was present. Additional data would be needed to verify whether the organization has a particular style that leans toward small team versus large team staffing.

Reliability Metric: ERRORS (SIT - FOC)

This graph shows the normalized total number of errors (critical, serious, moderate, tolerable, and cosmetic) found during systems testing until achievement of Full Operational Capability (FOC) (i.e. when the system was placed into production). The horizontal axis represents lines of code (new and modified); the vertical is the total number of errors.

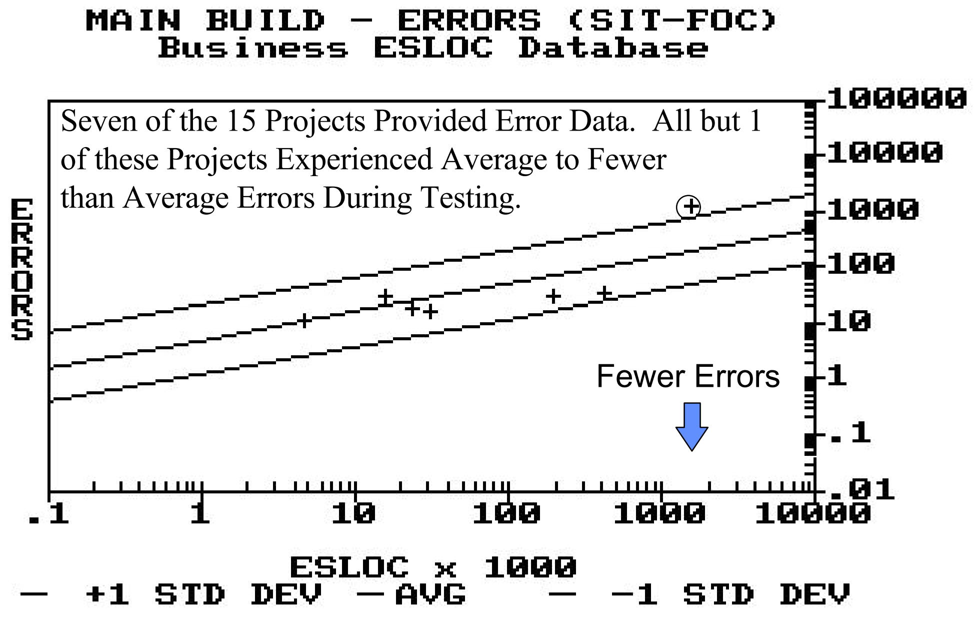

Figure 11. Errors (SIT - FOC) Trend.

Seven of the 15 projects maintained enough defect statistics to provide the number of errors found from the start of systems integration testing to the project completion. Six of these 7 projects also provided a breakdown of errors into 5 error categories; critical, serious, moderate, tolerable, and cosmetic.

The error trend in Figure 11 shows that 6 of the 7 projects reported total errors which were close to or less than the industry average. It must be emphasized that such comparable quality performance were achieved by the projects in about one fourth the time compared to industry.

The 1 project well above the average was 1 of only 2 projects above 1 million lines of code in size, and was comprised of several complex sub-projects. This project experienced a very lengthy requirements definition phase (see the Functional Design Schedule Trend - Figure 15) due to staff turnover involving business analysts and project managers. It also experienced a learning curve, both with the tools (internal development) and with the application type (external consultants). These environmental and complexity factors contributed to a higher number of errors. Unfortunately, this project did not have any breakdown of errors by category, so we couldn't get a sense of the distribution of errors by severity.

The other 2 large projects in the sample used small teams (see the Average Manpower trend - Figure 10) and expended less effort. The smaller team approach resulted in clearer communication amongst the staff during the design and construction. As a result, these projects experienced fewer errors as seen on this trend (1/2 to 1 standard deviation fewer than average).

One of the notable findings identified in this sample of error statistics (6 projects from 4 organizations) is that the errors identified in the NETRON/CAP development efforts are, on average, comprised of only 5% in the "serious error" (i.e. wrong answer) category, with less than 1% in the "critical error" (system crash) category. The sum of these 2 categories are 6%, versus a cumulative total of approximately 51% (serious and critical) from statistics in the QSM database. This suggests a good correlation between higher software quality and the re-use philosophy applied by the frame technology approach. A larger sample of error statistics would be required to verify the pattern seen in this small sample of projects.

The composition of the remaining error categories are 26% moderate , 24% tolerable, and 44% cosmetic.

Reliability Metric: MEAN TIME TO FAILURE

The Mean Time to Failure (MTTF) graph shows the average amount of time that the system runs between the observance of a defect during its first month of operational service. The horizontal axis represents lines of code (new and modified); the vertical axis is the average amount of time between discovery of defects in days. If your projects position above the average line, they are running without interruption for a longer than average time.

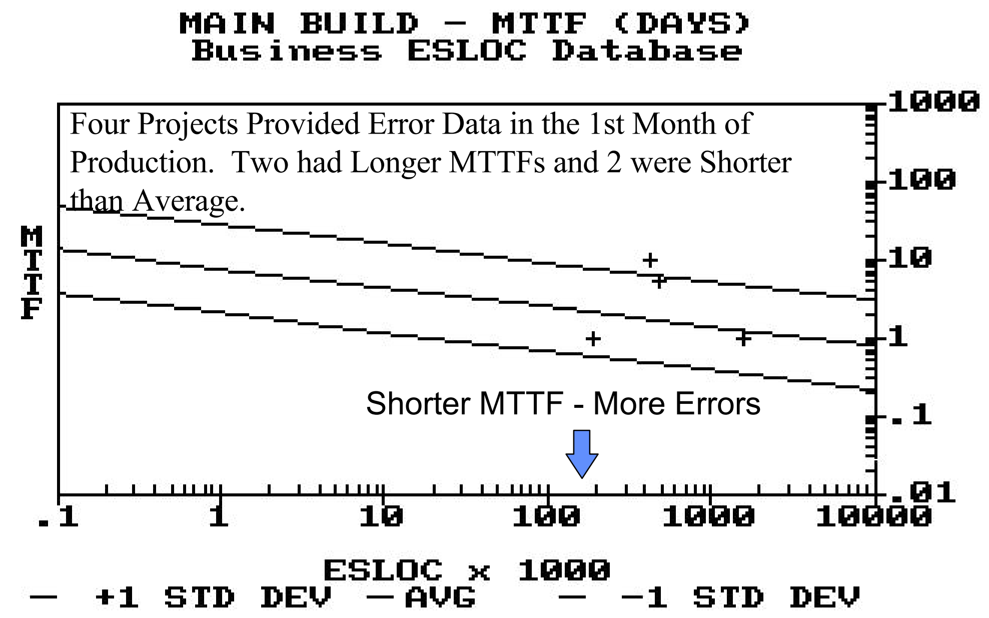

Figure 12. Mean Time to Failure Trend.

The 4 projects which contributed data on errors found in the first month of production were all in the larger system domain (200,000 to 1.5 million lines of code). The Mean Time to Failure (MTTF) is simply the average number of days that the system will run before an error occurs. Systems with a low number of errors remaining in the product when it is put into production will have higher MTTFs. Again, it should be noted that the competitive quality performance shown here was accomplished in roughly one fourth of the industry average schedule.

The project with the highest MTTF on this trend used a small team and had the fewest number of errors during development relative to its size, and the fewest number of errors remaining to be found after being placed into service, which resulted in a high MTTF. The small team used on this project contributed to a schedule that was somewhat longer than the other projects in the sample, but the resultant system was cost-effective and of truly high quality as evidenced by these 2 error trends.

The other project with a high MTTF (high quality) was the project we examined earlier (Project X) which used a very aggressive staffing. We do not have any of the error statistics for this project during development, nor any of the severity categories. However, assuming that the data provided in the first month of production is accurate, the high MTTF is an indication that the project staff had good skills, and that testing was well orchestrated and of enough volume to extract the bugs prior to production. It is much more common for large team projects to exhibit lower MTTFs, because more defects are created with the increased human communication complexity of larger teams.

The fact that this is not the case here, may also be attributed to the nature of Frame Technology. Re-using frames may allow you to create large applications with either large or small teams, and not necessarily pay the traditional penalty of higher defect rates normally associated with large teams. This may be due to the fact that frames are “encapsulated” and are manipulated in a structured and more disciplined, systematic approach. Again, we would need a larger data sample to validate this hypothesis. This sample, while valuable, is not entirely conclusive on this point.

As for the projects with shorter MTTFs (lesser quality), one reported low error rates during development. We suspect that this was a result of poor “defect extraction”, since they apparently surfaced later when the system was placed into service. In addition, only 6% of the errors found during development were critical or serious, while 70% were critical or serious in the first month of production. Had this system been more effectively tested and perhaps shipped a month later, the delivered quality would easily have been dramatically higher.

The other project with a lower MTTF was the largest project (rightmost on the trend). This was the project which experienced a high error rate during development (see the previous Error Trend - Figure 11). A significantly higher error rate during development often correlates to a lower MTTF during the first month of operation, because there are more total errors to be corrected.

Size Metric: AVERAGE CODE PRODUCTION RATE

This graph shows the average code production rate, measured in lines of code per calendar month. The horizontal axis represents lines of code (new and modified); the vertical is the lines of code per calendar month. If your projects position above the average line, your project teams are generating functionality and code at a faster rate.

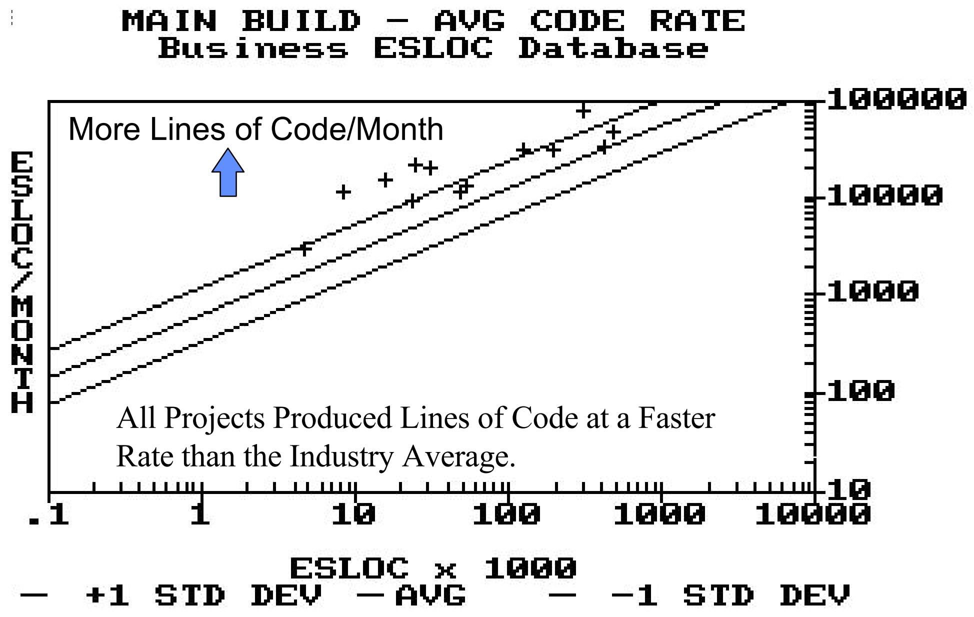

Figure 13. Average Code Rate Trend.

This graph is directly coupled to the project PIs and the very favorable Main Build schedule performance (Figure 8). All 15 projects had high PIs, and faster than average schedules. The code produced per month was therefore higher than average. The smaller projects with the fastest schedules had the highest code rates. Most of the larger projects had schedules closer to the average, and position closer to the average on this trend as well.

The code referenced in this trend is the code produced out of the NETRON/CAP tool. The rates are significantly higher than average due in part to the leverage provided by the re-use engine which takes advantage of the functionality that can be called upon from the frame libraries.

The hand-written instructions (what goes into the "front end" of the NETRON/CAP tool) were also counted. On average, 8.4 delivered instructions were produced (out of the "back-end") for every 1 line of hand-written code (NETRON/CAP SPC). The smaller projects (under 60,000 lines of code) achieved an average of 6.6 delivered lines per written line, and the larger projects (over 100,000 lines of code) achieved a higher average of 10.4 to 1.

Interestingly, the organizations categorized as first time users of the technology had the highest ratio of delivered instructions per written instruction, which is counter-intuitive. We expected the reverse to occur, namely that additional experience would be needed before being able to achieve this level of efficiency.

When this was scrutinized further, it appeared that the less experienced organizations received a "jump start" effect from more skilled and experienced NETRON consultants who provided frame engineering support on many of the first time projects. We would expect with a larger sample of more mature users (with standard corporate frames in place), that the ratio of code delivered by the tool as a function of written code into the front end of the tool would generally be higher compared to most novice users. This is equivalent to the long term pay-back which is expected of software re-use as a more mature infrastructure takes hold. However, it is important to note that all metrics assessed thus far (PI, schedule, cost, quality, and code ratios) show consistently good performance regardless of whether the user was new to the technology (or had consultant support).

NOTE: The largest project in the sample positioned above the upper end of the y-axis scale. This project positioned 1 standard deviation above the average, consistent with the other projects in the samp but is not visible on this trend.

Size Metric: PRODUCT PER PERSON-MONTH

This graph shows the average amount of code produced per unit of effort, measured in person-months. The horizontal axis represents lines of code (new and modified); the vertical is the lines of code per person-month. If your projects position above the average line, your project teams are generating more product per unit of effort.

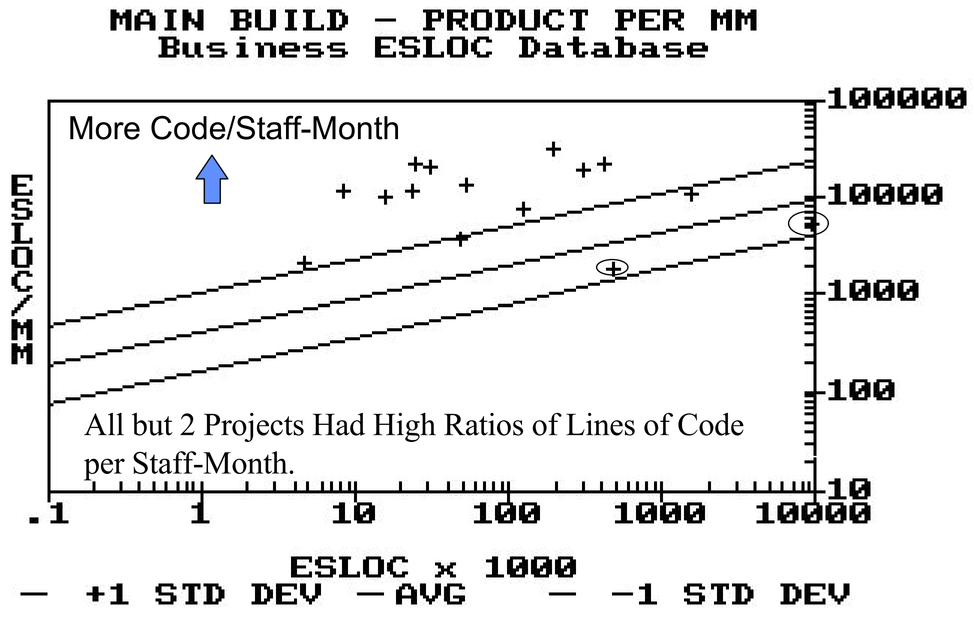

Figure 14. Lines of Code per Person-Month Trend.

This graph is directly coupled with the Main Build effort trend (Figure 9). The 2 larger projects which had higher than average effort expenditures due to the large teams used (in response to high schedule pressures), had lower ratios of lines of code per person-month of effort. Larger projects also expend more effort writing frames, which tends to reduce the ratio of product to effort, but should provide benefit for further re-use on subsequent projects.

All of the remaining 13 projects were very cost-effective due to the small teams used, and had line of code per person-month ratios well above the average (top 20%).

Note the volatility of the data when viewed by Lines of Code per Staff Month. This same volatility is evident on graphs for Function Points per person-month. From low to high, data for this metric has been observed to vary by as much as ten-fold. QSM's research over the last decade has revealed that the ratio (and the ratio of Function Points per person-month) are extremely unpredictable. They appear to be highly influenced by things like staffing decisions on projects, schedule pressure, and project to project complexity. Because of the near 100% failure rate when used for estimation, QSM recommends that they should not be used for planning purposes. If they are used for lack of more reliable methods, they should be used only with great caution and contingency.

Design Phase Metric: FUNCTIONAL DESIGN SCHEDULE

This graph shows the Functional Design phase (software requirements and high level design) for schedule in months. The horizontal axis represents lines of code (new and modified); the vertical is schedule (elapsed calendar months). If your projects position below the average line, they are spending less time in this phase of the project than normal.

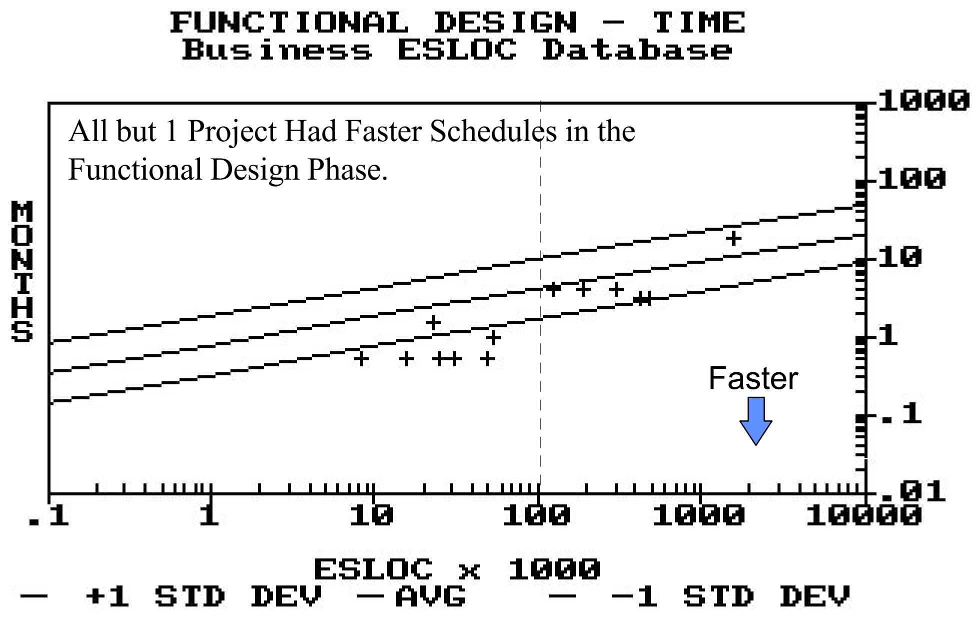

Figure 15. Functional Design Schedule Trend.

This trend shows that all but 1 of the projects assessed consistently spent average to less time than average in the Functional Design phase. This phase is typically driven by the complexity of the software requirements and the thoroughness applied to developing a sound high level design.

The smaller projects in the sample (less than 60,000 lines of code) tended to have very short Functional Design phases (average of less than 1 month). Smaller projects typically require less time to develop a stable set of requirements and high-level design than larger projects. The small projects which were more than 1σ faster than average also maintained a competitive schedule in the Main Build phase. This indicates that these projects did not just rush to the construction phase, but rather only required a short Functional Design.

The larger projects (over 100,000 lines of code) were still faster than the average, but generally spent more time in this phase to stabilize requirements and high-level design (average of 6 months). The largest project (rightmost on the graph) reporting data in this phase (1.5 million lines of code) spent about 1/2 standard deviation longer in this phase. This was the project which was impacted by staff turnover (Business Analysts and Management Staff). The team also had to absorb a learning curve with both the technology and the application. However, the Main Build schedule for this project was very competitive. It was even shorter than the Functional Design. However, the defects were higher when the system was put into service, so in retrospect it appears that the Main Build may have been rushed. (The error rates were higher than average for this project both during and after production - see Error Trends).

Design Phase Metric: FUNCTIONAL DESIGN EFFORT

This graph shows the Functional Design effort in person-months. The horizontal axis represents lines of code (new and modified); the vertical represents effort (measured in person-months). If your projects position below the average line, they are using less effort and costing less than the average for this phase of the project.

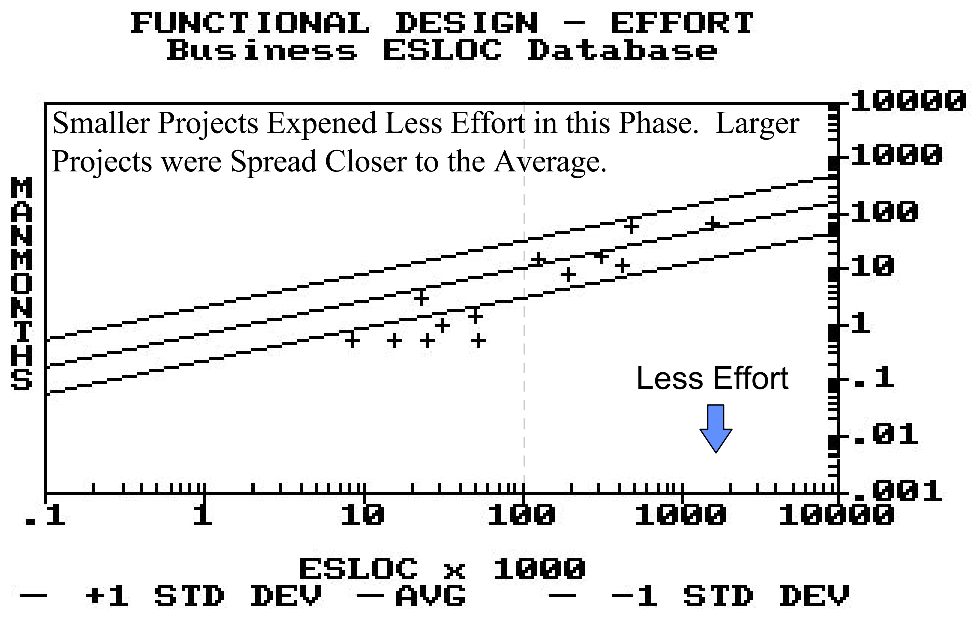

Figure 16. Functional Design Effort.

The smaller projects in the sample all used small teams in this phase (average of 2 people) and had consistently lower than average effort expenditures (average of 1 person-month).

The larger projects were all spread within 1/2 σ of the industry average, with an average effort expenditure of 29 person-months (sample average of 9 people). Only 1 project had a significantly higher effort expenditure. This project staffed up very rapidly and had an average team size roughly 5 times higher than the industry average, with an effort expenditure almost twice the sample average. This project followed the same aggressive staffing throughout the project, and subsequently had good schedule performance in all phases (which is usually the intent of larger teams), but was more expensive than the average as a result.

The largest projects in this sample also tended to follow a waterfall approach in their developments, rather than an iterative build, which was more typical of the smaller projects. This may have contributed to longer schedules and higher than average effort in the Functional Design phase as compared to the smaller projects in the sample.

Analysis Phase Metric: FEASIBILITY STUDY SCHEDULE

This graph shows the Feasibility Study phase (cost and technical feasibility, business case and implementation plan development) for schedule, measured in months. The horizontal axis represents size measured in lines of code (new and modified); the vertical is the metric for schedule (elapsed calendar months). If your projects position below the average line, they are spending less time in this phase of the project than normal. Above the average line, they are spending more time than normal.

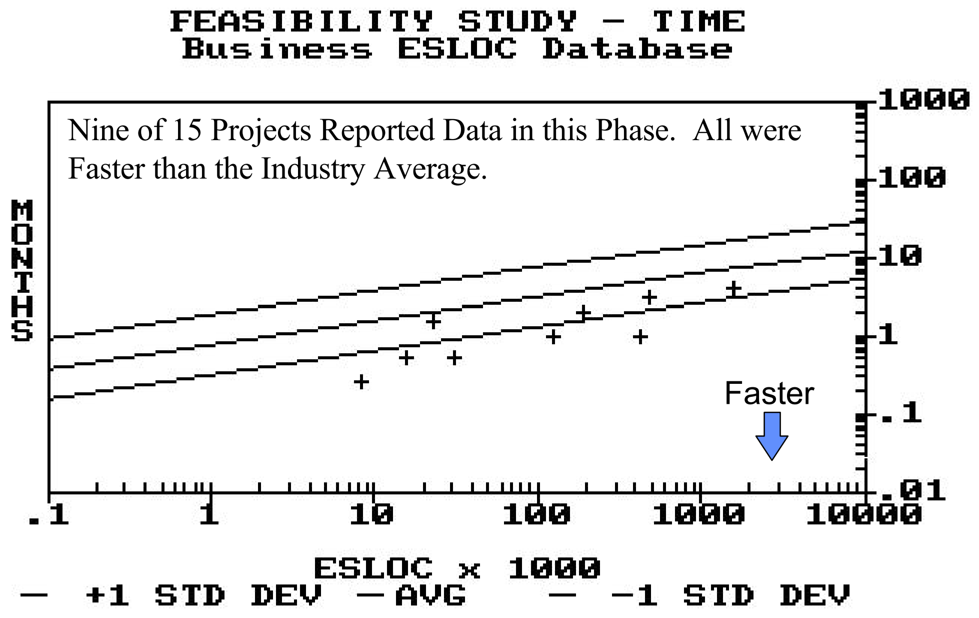

Figure 17. Feasibility Study Schedule Trend.

The duration of the Feasibility Study phase tends to be driven mostly by the business environment. Nine of the 15 projects contributed data in this phase. Five of the remaining 6 projects were small enough that an actual phase of the lifecycle was not required to determine the business case. The 9 projects which did report on this phase, all positioned faster than the industry average for schedule performance. This phase was typically very short in duration (average of 1.5 month for the sample).

Schedule performance in this phase may be linked to the prototype capabilities of re-usable frames and architecture templates, but there was insufficient data available to be conclusive. However, the data shows clearly that Main Build schedules were not negatively impacted by the short duration of the feasibility study phase.

Analysis Phase Metric _ FEASIBILITY STUDY EFFORT

This graph shows the Feasibility Study effort in person-months. The horizontal axis represents lines of code (new and modified); the vertical represents effort (measured in person-months). If your projects position below the average line, they are using less effort and costing less than the average for this phase of the project. If they are above the average, they are using more effort for this phase.

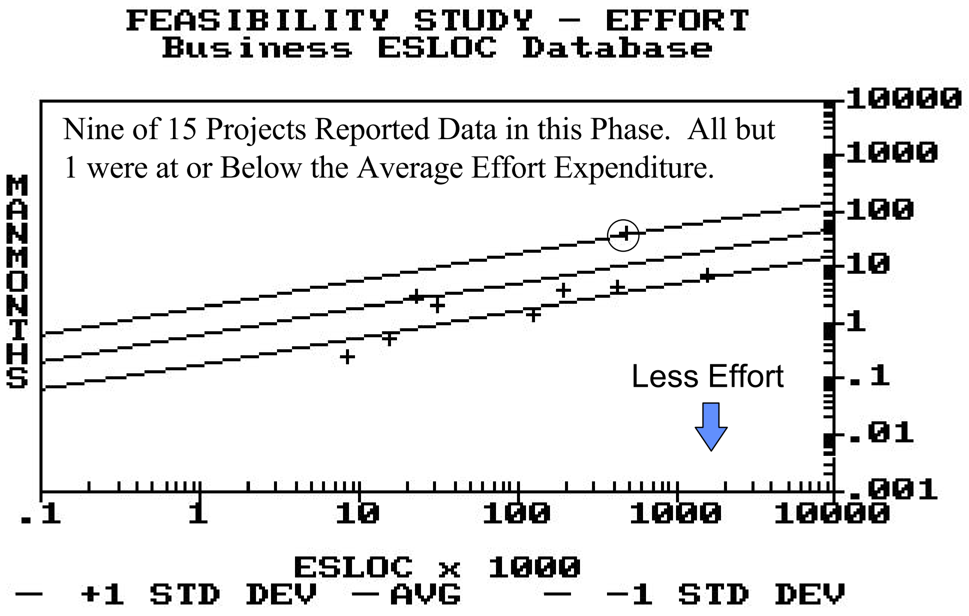

Figure 18. Feasibility Study Effort Trend.

All but 1 of the 9 projects reporting data in this phase had lower than average effort expenditures. Team sizes are typically very small in this early phase of the lifecycle. The one project at +1 σ from the average had a high staff level from day one and was highlighted on many of the previous trends for higher effort expenditure but good schedules throughout all phases of development and test.

Project Composition Analysis

Assessment of Development Attributes

Each of the contributing organizations provided project data which identified positive and negative attributes of the development environment. These qualitative areas (i.e. soft factors) include team skills, project constraints, complexity drivers, and other factors unique to each application.

In this section we'll describe these characteristics in aggregate form, and highlight any common attributes of projects in the sample.

SCHEDULE AND RESOURCE DISTRIBUTION

All projects reported data on the Main Build (detailed design, construction, test and validation) phase of development. Thirteen of 15 provided data for the Functional Design Phase (requirements definition and high level design), and 9 of 15 for the Feasibility Study (business and technical feasibility assessments). The breakdown of the average time and effort spent in each phase is shown in Figures 19 and 20.

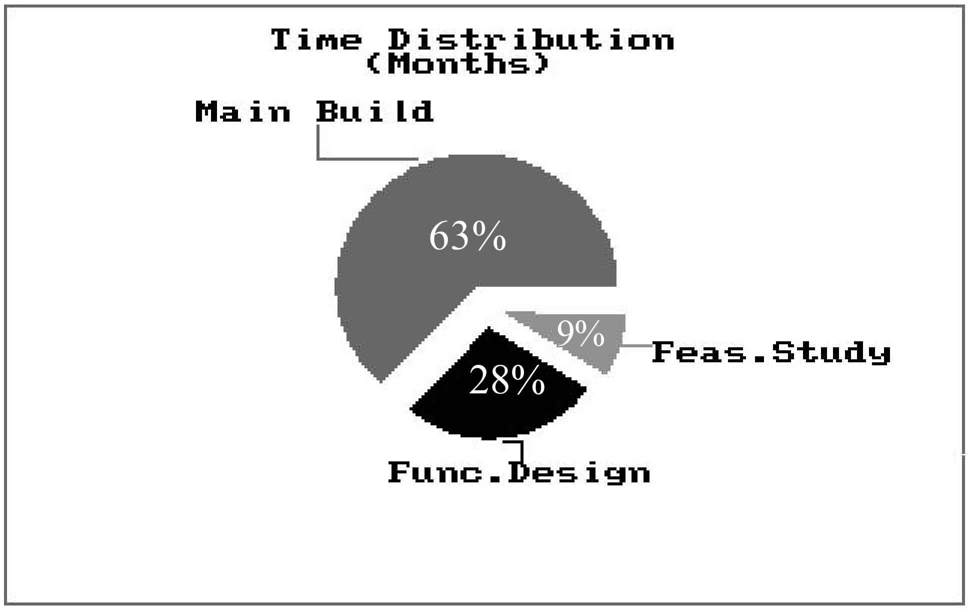

Figure 19. Schedule Breakout by Phase.

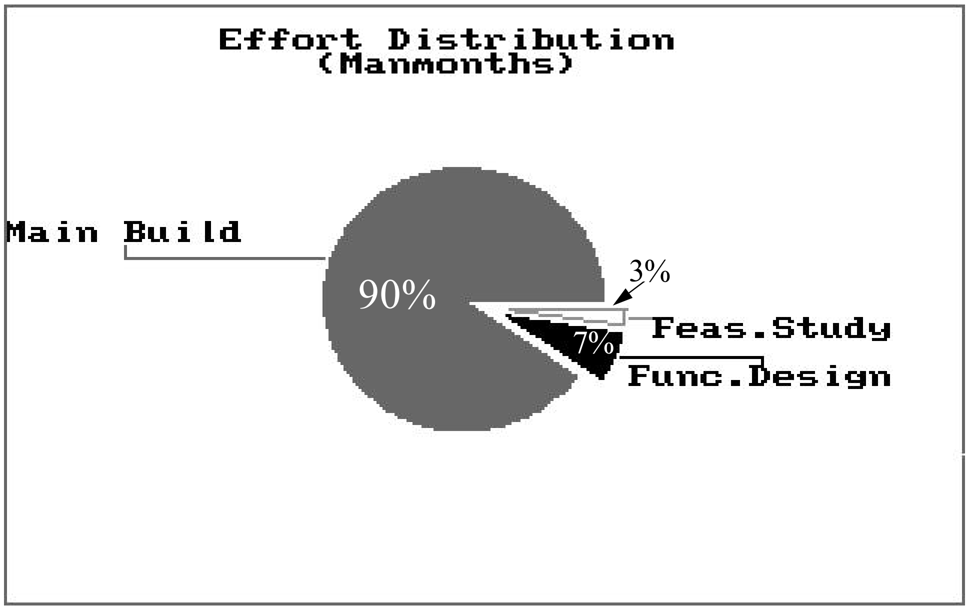

Figure 20. Effort Breakout by Phase.

The percentages listed in Figures 19 and 20 represent the aggregate behavior of the sample projects for each phase of development. The overall pattern has most of the time and effort (63% and 90% respectively) expended in the Main Build phase. This is close to the QSM industry average for schedule (industry average - 57%), but considerably higher than the industry average for effort (industry average - 72%).

The high percentage of effort in the Main Build was strongly influenced by the 2 large projects in the sample. These projects delivered 1.5 million and 9.3 million lines of code, and made up 85% of the total Main Build effort expenditure across the entire sample. With these 2 projects omitted from the sample, the schedule and effort breakout by phase is within 5% of the industry averages for all phases.

Thus, for all projects under 1 million lines of delivered code, the breakout of time and effort expended by phase is very similar to the industry averages (with 5% more time and 5% less effort spent in Main Build). This suggests that the reduction in effort and cycle time posed by the re-use methodology is fairly consistent for all 3 phases, and results in a relative reduction for each of them.

COST OVERRUN AND SCHEDULE SLIPPAGE

Only 3 projects of the 15 reported schedule slips from the original plan, and only 2 reported effort overruns.

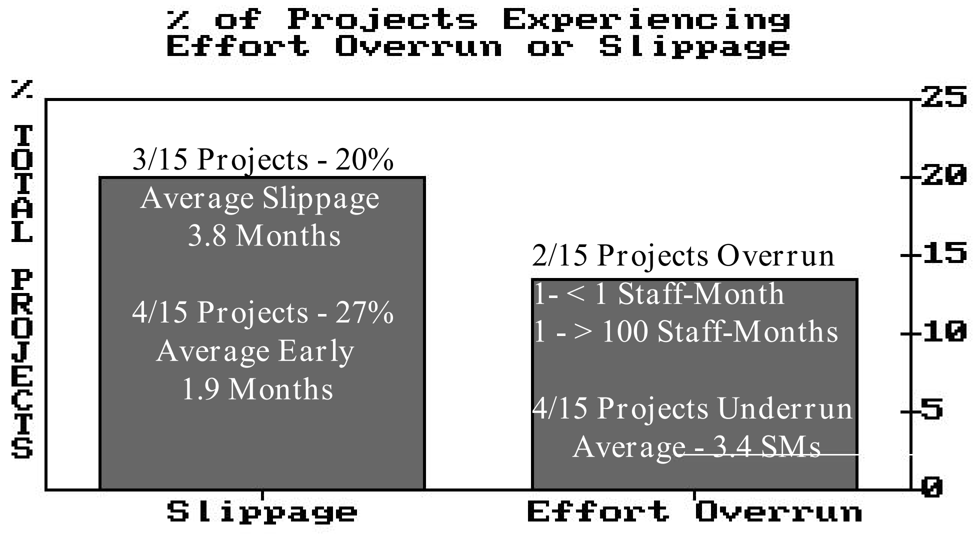

Figure 21. Cost Overrun/Schedule Slippage History.

The schedule slippage reported on 3 of 15 projects averaged 3.8 months. In contrast, 4 of 15 projects reported being ahead of schedule by an average of 1.9 months. The remaining 8 projects were either on schedule or the data was not available from the original plan.

Of the 2 projects which reported effort overruns, 1 was very small in code size (second smallest in sample), and had an overrun of less than 1 person-month. The other was much larger (second largest in code size) and had an overrun of over 100 person-months (in this case, poor project estimation seems to have existed independent of the high productivity achieved by re-use).

Four of the 15 projects reported effort actuals averaging 3.4 person-months below plan.

In comparing the schedule and effort performance for this sample to the QSM I/S averages (a much larger sample), the schedule and effort slippage and overruns are similar (in terms of both frequency, and the average slip or overrun in months and person-months). However, this sample of NETRON/CAP projects has a greater frequency of both schedule and effort underruns than the average. Only 2% of industry I/S projects reported schedule underruns and only 6% reported effort underruns. In this relatively small sample of 15 NETRON/CAP projects, both schedule and effort underruns occurred in 27% of the projects. This may be due to the unexpectedly positive schedule and effort performance exhibited on the majority of projects by first time users of the technology. In fact, 2 of the 4 organizations reporting project underruns were first time users, and 2 were newer intermediate users.

Figure 22 compares the actual PI achieved, against the PI implied by the original plans of the 3 projects which reported slippage or overrun. These projects had original plans which aimed for PIs 5.3 higher than what was achieved. This was unrealistic. In fact, the actual PIs achieved averaged 25.8, which is 8.9 PIs above the industry average. Therefore, what is reported as slippage or overrun, is also a function of aggressive and sometimes unreasonable plans.

The project with the 100 person-month overrun (Project 13) is a good example of original plans that were blatantly unrealistic. This project was over a million lines of delivered code, was comprised of multiple sub-projects, had staff turnover in key personnel, and was a first time NETRON/CAP development. The PI achieved on this project was very credible, but the original plan (based on the schedule and effort overrun) was more than 5 PIs above what was achieved, and higher than any PI exhibited in this sample.

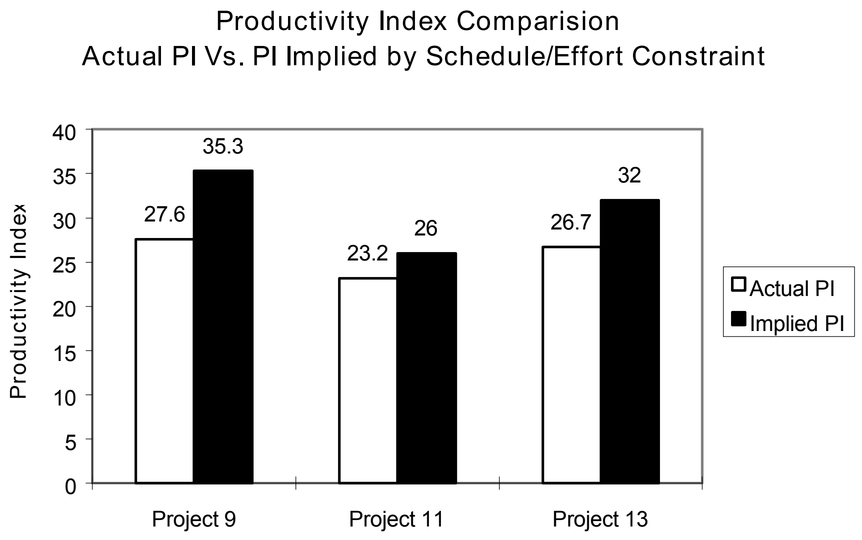

Figure 22. Actual PIs Vs. Planned PIs

CONSTRAINTS

The figure below shows the significant areas where the NETRON/CAP projects experienced constraints.

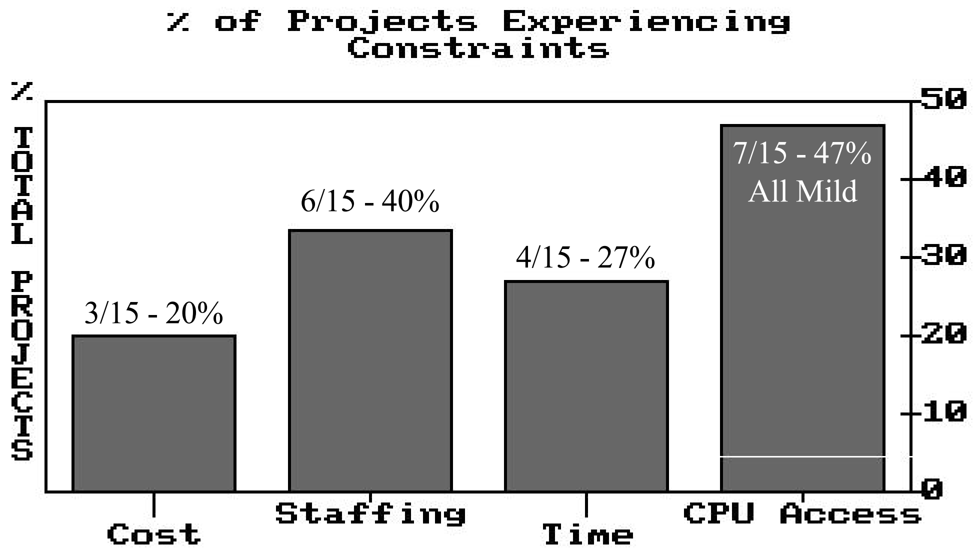

Figure 23. Project Constraints.

Of all the constraints placed on the projects in this sample, staffing was the most severe. Six of 15 projects (40%) reported Main Build staffing limits. The average staff constraint was 3 people, and all but 3 of the largest projects in the sample had peak staffs no greater than 6 people. This helped to contain effort expenditures. However, it is more typical for small team projects to have longer than average schedules. This was not the case in this sample. The re-use solution provided significant functionality with faster than average schedules, while maintaining a small team approach on the majority of projects.

Three of 15 projects (20%) reported Main Build cost constraints averaging $1.2 Million, and 4 of 15 (27%) reported schedule constraints averaging 7.8 months. While only 3 projects reported cost constraints, 50% had either staffing limitations (equivalent to cost) or a cost constraint.

The number of schedule constrained projects was less than expected due to the lack of data in this area. 80% of projects gave indicators of project success relative to on-time, overrun, or underrrun performance, but they did not indicate what the original performance targets were.

Seven of 15 projects also reported computer constraints. These were all mild, and no projects indicated that access to development hardware or limitations in processing power were a significant detriment to development progress.

ENVIRONMENTAL FACTORS

The figure below shows the average percentage of staff turnover, requirements change, memory occupancy and real time code for the NETRON/CAP projects.

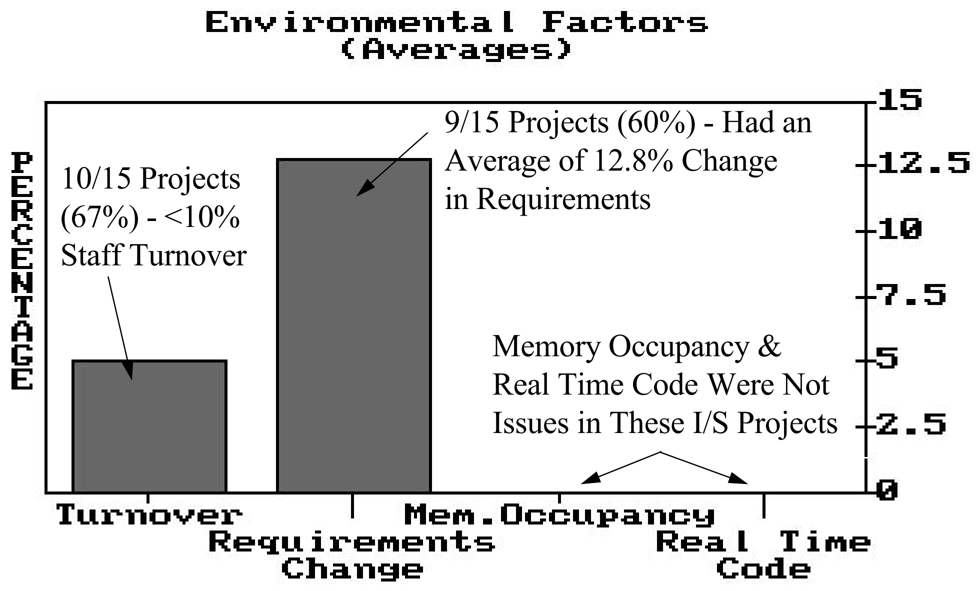

Figure 24. Environmental Factors.

Staff turnover was below 10% of the total team size for 10 of 15 projects (67%). Three of the remaining projects had staff turnover rates above 10% and experienced PIs 2 values below the overall sample average (24.1 versus 26.2). Turnover in staffing (switching projects or organizations) is typically a contributing factor to PI degradation, especially on small team efforts which are more susceptible to the loss of 1 or 2 key people.

Requirements change was measured as a rough percentage from the start of Main Build (when the software requirements are theoretically baselined). Nine of 15 projects reported requirements change which averaged 12.8% from the original baseline (the remaining 6 did not report any data). QSM industry statistics show that a figure of 20% or higher can degrade the PI by 1, 2, or more values. The low level of requirements change in this sample may have helped the PIs for these projects as a whole.

The Memory Occupancy and Real Time Code categories were not applicable to this set of projects. These are more typical of complexity drivers on engineering applications which operate in a more constrained computing environment.

TEAM SKILLS

The figure below shows the breakdown of skills across the development teams used on the NETRON/CAP projects.

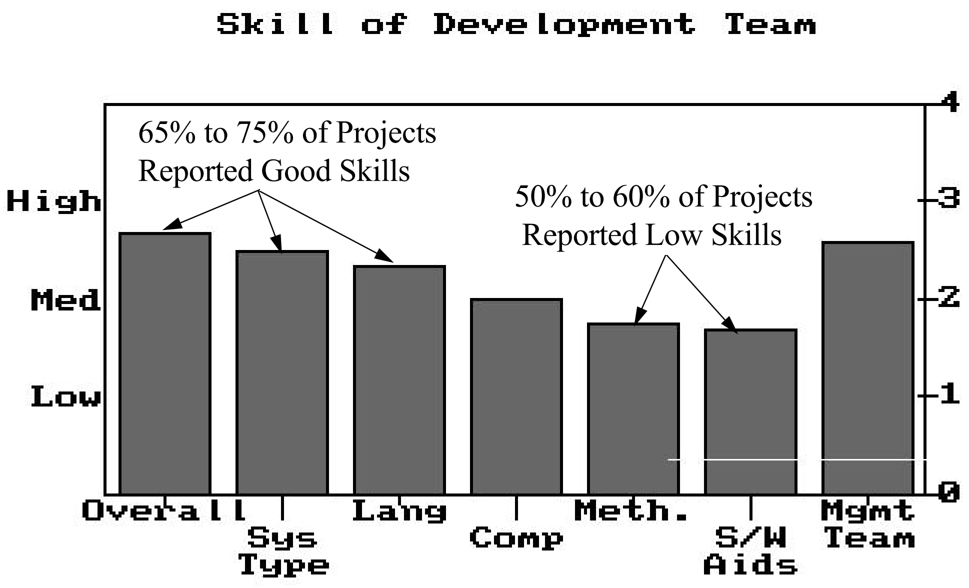

Figure 25. Development Skills.

Each of the contributing organizations was asked to provide an indication of team skills in a number of categories including experience and familiarity with the application type, the development language, development computer, the project methodologies, the software aids (tools and utilities), and the level of management experience. An answer of "low" corresponded to an average of less than 1 year experience, "medium" corresponded to an average of 1 to 3 years experience, and "high" was greater than 3 years experience.

As shown in Figure 25, the overall category (combination of the other 6 categories), the familiarity with similar systems, and the management team experience, had the highest average skill ratings. Extensive knowledge of the business applications, and management experience generally help to raise the Productivity Index, especially on small team projects which are more sensitive to individual contributor skills.

Language experience was just above medium, while experience with the software aids (tools and utilities) averaged below medium. First time users of NETRON/CAP all indicated low to medium experience in these 2 categories, with a larger number of low responses in the software aids category. Because there are a minimal number of commands to master within the NETRON/CAP frame command language, development language did not appear to be a large factor in the learning curve. In addition, first time users typically had NETRON consulting support, which increased their average experience with the frame command language. The tools and utilities experience categories appeared to be influenced by additional factors outside of NETRON/CAP.

SYSTEM FEATURES AND DESIGN COMPLEXITY

Many factors contribute to the complexity of a given software project. The figures below show a breakout of several key system features and complexity drivers identified in the NETRON/CAP project sample.

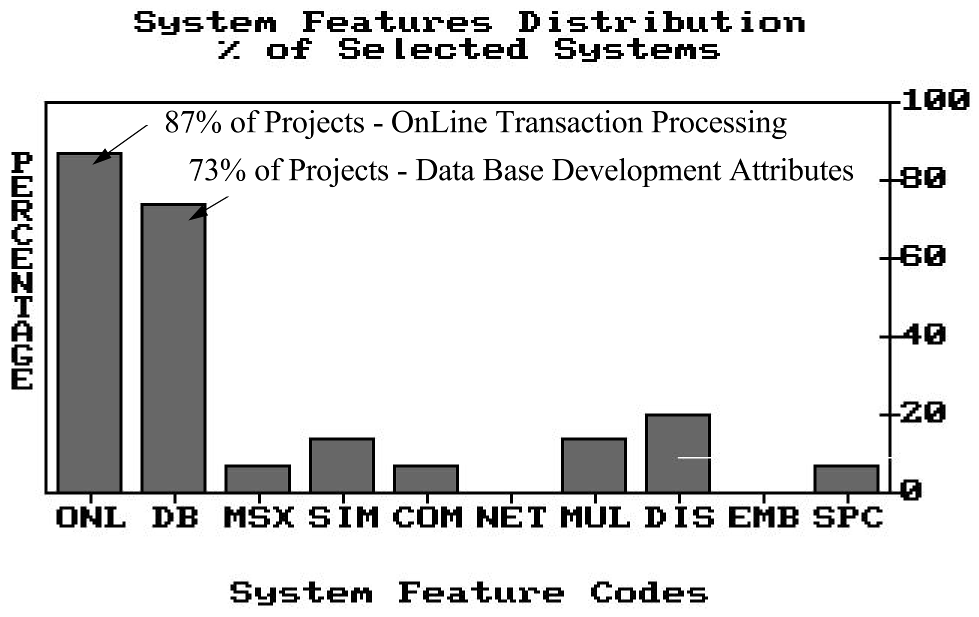

Figure 26. System Features.

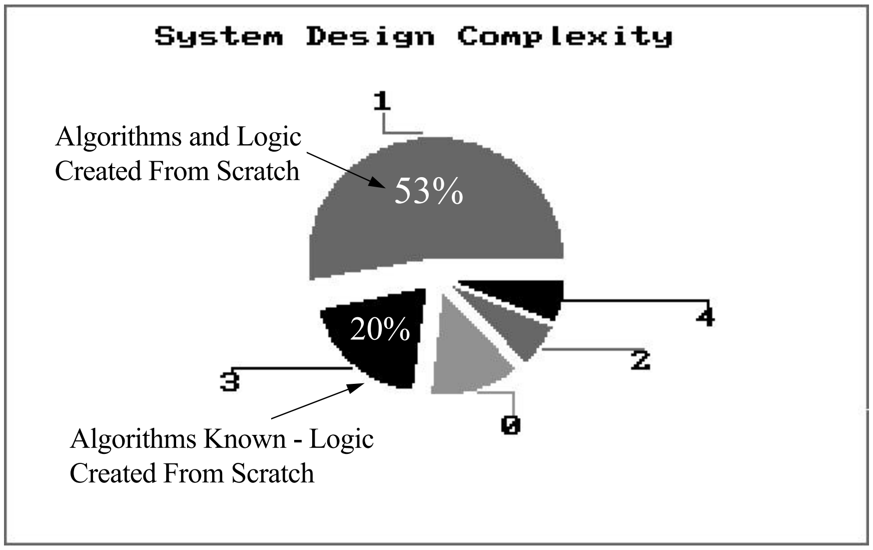

Figure 27. System Design Complexity.

Like most Information Systems, the predominant system features (Figure 26) were On-Line Transaction Processing and Database (87% and 73% of projects reported these features). In addition, 7 of the 15 projects had other, more complex features including message switching, simulation, communications, multi-processing, and distributed architectures.

Each of the projects also provided an indication of system design complexity using a scale from 0 to 6, with 1 being the most complex design (all new algorithms and logic), and 6 being the least complex (conversion effort). 53% of the projects indicated the highest complexity level of 1. Although the majority of projects reported completely new algorithm and logic design, software re-use was still very significant. This apparent paradox occurs because what is "new" to the application design, is very often existing in the frame libraries, and may be re-used via modification. In addition, any frames created expressly for the application become a resource to subsequent projects.

The next highest percentage was complexity level 3 (algorithms are known, all program logic is created from scratch). The remaining categories were: level 0 (13% - unknown, data not entered), level 2 (7% - algorithms and logic mostly created from scratch), and level 4 (7% - rebuild with up to 30% new functionality).

Splitting the project sample above and below the sample average PI (26.2), results in 6 projects below the average, and 9 above. Of the 6 projects below the average PI, 5 were among the most complex from a design and feature complexity perspective. Of the 9 projects above the PI average, only 1 project had a high complexity rating of 1 combined with non-traditional system features. This is a useful observation relative to future planning of NETRON/CAP projects. Complexity of system features and design, as well as the other environmental factors (skills, requirements stability, expected turnover, etc.), can influence the PI which is feasible for a new effort. These factors should be carefully assessed prior to making project commitments.

Glossary of Terms

Effort. The resource expenditure on a software project as expressed in person-months. Determined by summing the staff applied during each calendar month of a software project. Includes all development staff: analysts, designers, programmers, integration and test-team members, quality assurance, configuration management, documentation, supervision, and project management.

Errors. An error in analysis, design, coding, or testing that leads to a fault in a program that causes a failure in the sense that program operation departs from the pattern specified or, in the case of an error in the specifications themselves, departs from the desired pattern. More precisely, a deviation from the required (or desired) output by more than a specified tolerance. In popular usage, a “bug”.

ESLOC. Effective Source Lines of Code. This is the new and modified delivered source code from an application excluding comments and blanks. This includes executables, data declaration, and compiler directives, and is a count of logical instructions independent of the physical format in which they appear.

Feasibility Study. An early phase in the software lifecycle to develop a complete and technically feasible set of requirements for the system and formulate the top level approach for their implementation. This phase typically includes the development of a Statement of Need, Business Case, Marketing Specification, or Feasibility Study Report.

Frame. An adaptable software component, or sub-assembly, used in the automated assembly of source modules - objects, programs, subroutines, texts in any language. Frames contain data structures and methods which other frames may adapt - select, modify, extend, delete, iterate - to their needs. Each module is assembled from a hierarchy of frames, represented as a parts explosion diagram.

Frame Technology. An advanced form of object-orientation, an automated assembly process which manufactures software modules from adaptable, generic components called frames.

Functional Design. A phase of the software development process prior to the Main Build to develop the software requirements of the system, and a technically feasible design within the scope of these requirements. Typical products of this phase include a Functional Specification or Software Requirements Specification, a High-Level or External Design Document, an Interface Design Specification, Test Program Plans, and Management Plans. This phase is typically overlapped with the Main Build, and may be repeated in an incremental development lifecycle.

Function Point. A method of measuring the functionality of a proposed software development based upon a count of inputs, outputs, master files, inquiries, and interfaces. The method may be used to estimate the size of a software system as soon as these functions can be identified. It can also be used to describe the size of a system after project completion.

Main Build. (Also known as Main Construction) A phase in the software lifecycle to produce a working system that implements the specifications and meets the requirements in terms of performance, reliability, and interfaces. Begins with detailed logic design, continues through coding, unit testing, integration, and system testing, and ends when the system is fully operational and can be moved into production.

MTTF. Mean Time to Failure. The average time between discovery of errors in the system, measured over a specified time.

NETRON/CAP. An object-oriented construction tool set, and an industry standard Cobol frame library. From high level specifications, NETRON/CAP assembles Cobol source modules; compiles and links them into executable systems for multiple client/server/GUI environments.

PADS. Productivity Analysis Database System. An automated reference data base tool which allows for the capture of project data, calculation of process metrics, and benchmark positioning for schedule, cost, and quality performance against industry trends.

Productivity Index. (PI) A management scale from 1 to 40 which encompasses a macro measure of process efficiency and project complexity. This number can be calculated for completed projects and for current plans using the project size, elapsed time to build, and effort expenditure. Each integer change in the PI scale represents approximately a 10% difference in schedule performance, and a 27% difference in cost performance.

Schedule. Time of each phase of a project measured in calendar months from the start.

SLIM. (Software LIfecycle Management) An automated decision modeling tool which predicts software project costs, schedule, and quality. Dynamic risk management can be performed and management plans assessed based on the governing project constraints. Inputs to the model include process productivity, system size, application complexity, and project staffing.

SPC. (SPeCification frame). At the top of each frame hierarchy sits a specification frame, so called because it specifies a module's unique details and its main sub-assembly frames; it is the module's master blueprint. There is one SPC per module and it is not normally reusable.

Standard Deviation. A measure of the amount of variability about the mean of a set of numbers. Specifically, the square root of the sum of the squares of the deviations from the mean divided by the number of times in the set, less one. Also known as the Greek letter, sigma. 68.26 percent of the cases lie within plus and minus one standard deviation of the mean; 95.44 percent, within plus and minus two standard deviations; 99.74 percent, within plus and minus three standard deviations.

Bibliography

Bassett, Paul G., "Frame Based Software Engineering", IEEE Software, Vol 4, no. 4 (July 1987); also reprinted in "Software State-of-the-Art; Selected Papers", edited by Tom Demarco and Timothy Lister, Dorset House, 1990.

Bassett, Paul G., "Engineering Software for Softness", pp 24-38 in American Programmer Vol 4. no.3 (March 1991).

Martin, James, "Bassett Frame Technology", chapter 16 in "Rapid Application Development", Macmillan, 1991.

Putnam, Lawrence H., and Mah, Michael C., "Is There A Real Measure for Software Productivity?", Programmer's Update, June 1990.

Putnam, Lawrence H., "Measures for Excellence - Reliable Software, On-Time, Within Budget" © 1992 Prentice Hall/Yourdon Press

Putnam, Lawrence H., "Software Re-Use in Japan", published as Chapter 8, Quantitative metrics, Technology Transfer International, © 1992.