The 2011 COSMIC/ISBSG Global Benchmarking Initiative

Dear Reader,

With this letter, we would like to invite you to contribute to the '2011 COSMIC/ISBSG Global Benchmarking Initiative'.

With the publication of version 3.0.1 of the COSMIC Functional Size Measurement Method in 2009, and its updated approval as an ISO standard earlier this year, the method reached a new level of maturity and stability. Many organizations worldwide value the method for measuring performance, estimating and benchmarking their IT projects.

At this moment, well over 400 projects measured with COSMIC are present in the ISBSG repository. Although this is a significant number that would fulfil most of the analysis and benchmarking needs of most organizations, we feel that a new initiative could be very valuable. Today, there are so many users that we should be able to gather enough data to establish much more refined benchmarks, thus adding much greater value to those submitting data and to the users' community in general.

The task for your organization should be really simple, namely to enter data on as many projects as you can on the COSMIC/ISBSG 'Concise Data Collection Questionnaire (CDCQ)' (attached document). Data should be submitted before mid-September, to ensure it will be included in the next release of the ISBSG Benchmark and in the next Benchmarking Report. Assuming the data for, say, 5 projects are readily available, it would require just a few hours to enter the data via this concise questionnaire.

Please note that there is no limit to the number of projects whose data you can submit – the more data you can submit, the more value your organisation will add and receive. We would also ask you to consider that your organization has already profited from the free availability of the 'open' COSMIC method which was developed as a non-commercial effort for the general benefit of the software community. By participating in this initiative, you will also be contributing to this common good by helping to increase the value of the method, which should aid its increasing adoption.

Each organization participating in this initiative will gain significant benefits, namely:

- A **private report** from the ISBSG for each, single project submitted, comparing its performance against similar projects in the ISBSG database. (This report will help you identify opportunities for performance improvement.)
- In addition, for those submitting at least 5 projects, a **free copy of the complete benchmarking report** analysing all COSMIC-measured projects in the ISBSG database as at end of 2009, i.e. before your new submission(s). (A free, abridged version of this report – i.e. with graphs and plots, but no real numbers - is attached for your reference; the complete report is valued about 100 USD).
- Further, organizations that submit at least 10 projects will receive **a free copy of the updated benchmarking repor**t, analysing the newest COSMIC sample, by end of 2011. (When officially issued, this report will not be free as well as the previous one to non-submitters – price to be announced.)
- Finally, organizations that submit more than 15 projects within the timeframe of this initiative will also receive a **free copy of the book "COSMIC Function Points"** by Professors Dumke and Abran, 2010(value about 100 USD).

Note that data submitted for the projects should receive a data quality rating of 'A' or 'B' from ISBSG in order to qualify for these last two awards. As an aid to help you achieve this quality level, we also attach the latest COSMIC 'Guideline for assuring the accuracy of measurements'. While not mandatory, this guideline will also help you understand the inner quality of your measurement process in your organization.

The COSMIC and ISBSG organizations guarantee that any project data you submit will be anonymized so that no-one will be able to link that data to your organization. There is no risk of embarrassment to your organization or loss of private data as a result of submitting data to the ISBSG with this initiative.

Please note this is a global initiative. This letter is being sent out to known COSMIC users as well as to functional sizing practitioners worldwide. If you know of other users potentially interested by this initiative, please feel free to share this invitation with them.

We hope this proposal appeals to you and that your organization will agree to participate in this valuable exercise. Meanwhile, if you have any questions, please do not hesitate to contact us as below.

Yours sincerely,

Harold van Heeringen (harold.van.heeringen@sogeti.nl)
Luca Santillo (luca.santillo@gmail.com)

COSMIC Intl Advisory Council
Benchmarking Committee

The Performance of Real-Time, Business Application and Component Software Projects

An analysis of COSMIC-measured projects in the ISBSG database.

This version of the report contains all graphs and tables but almost no measurement results. Its purpose is to show the reader the types of data and analysis that are available in the full report of September 2009, obtainable from www.isbsg.org

April 2011

Abstract

This report provides an analysis of the data in the ISBSG database with software sizes measured using the COSMIC Functional Size Measurement method, including all data collected in the COSMIC/ISBSG Benchmarking Initiative up to May 1st 2009.

Performance data on 61 projects from the domains of real-time applications and of software components, and on 298 projects from the domain of business application software, both new developments and enhancements, are analyzed to produce benchmarks suitable for performance comparisons and for project estimating.

The report is an updated version of a report dated „June 2009" with additional analyses and a summary of findings and conclusions.

Published by:
The Common Software Measurement International Consortium („COSMIC"), and
The International Software Benchmarking Standards Group („ISBSG")

Compiled and edited by Charles Symons, Joint Project Leader, COSMIC

Reviewed by:
George Ansell, ISBSG
Peter Hill, ISBSG
Harold van Heeringen, Sogeti, Netherlands
Luca Santillo, Agile Metrics, Italy

Contents

1. Introduction

This report presents an analysis of the data on projects in the ISBSG database with software sizes measured using the COSMIC Functional Size Measurement (FSM) method, including all data collected in the COSMIC/ISBSG Benchmarking Initiative up to May 1st 2009. Data on projects from another source are also included for certain analyses. The aim is to produce data that will be useful for benchmark comparisons and for input to project estimating methods, such as described in [1]

The projects were completed in the period 1999 – 2009.

This report is an updated version of an „initial" report on the same data, dated June 2009. The principal changes are as follows.

- A deeper analysis of the recorded „effort" data (see Appendix A) leads to some improved benchmark figures, such as the distribution of effort by project activity for business application projects. Readers who will be using these benchmark data in practice, e.g. for project estimating, are strongly advised to read Appendix A so that effort data are correctly interpreted
- Some data are added on the productivity of business application projects that developed software to execute in batch mode compared with on-line mode.
- The analysis of project effort data is separated into a new Chapter 4 and some new analyses added
- A new Chapter 5 includes some data comparing the characteristics of COSMIC measurements of functional size for different types of software
- A new Chapter 6 provides comments and some data on comparing benchmark figures where the functional size of delivered software has been measured using the COSMIC, IFPUG and MkII FPA methods
- A new Chapter 7 on Summary Findings and Conclusions has been added
- Some minor arithmetic errors have been corrected. One major correction has been made to the charts (now Figures 14 and 15) showing the trends of productivity of enhancement projects per size band. We apologise for these errors in the „initial" report.

The additions and changes from the „initial" version are described in more detail in Appendix B.

Readers are assumed to have a broad understanding of the COSMIC method for measuring a functional size of software. For those needing an overview of the method, go to www.cosmicon.com, or obtain the „Method Overview" document [2].

After this Introduction, the report has two main Chapters containing the principal benchmark data analyses. Chapter 2 concerns the analysis of the data on 62 projects that have been classified as being concerned with real-time applications, with re-usable software components, and with some miscellaneous types of software. Chapter 3 concerns the analysis of 272 ISBSG-recorded projects and 26 projects from another source, all from the domain of business application software.

The reasons to distinguish these two main categories of software are that it is anticipated that the performance data for these projects will vary with the software domain, and that the audiences for these two main categories are different.

The main performance parameter analysed is that reported by the ISBSG to establish benchmarks for all projects in its database, namely the „project delivery rate", or „PDR", defined by:

PDR = Effort (Work-hours) / Size (COSMIC Function Points)

N.B. This parameter is the inverse of the more conventional „productivity" parameter, so the lower the PDR, the higher the productivity, and vice versa. Graphs in the report that show individual project performance as scatter diagrams have a horizontal x-axis labelled „Size (CFP)", with the vertical axis labelled „Effort (Work-hours)" or „Elapsed months". On all these graphs, individual projects represented by dots that appear higher up the vertical axis (or above a trend-line) have a poorer performance i.e. lower productivity (=higher PDR) or lower speed respectively, than projects shown by dots lower down or below a trend line.

A few graphs show conventional project „productivity" (measured in CFP/work-month) and project „speed" (CFP/elapsed month) versus a parameter such as size of the delivered software. For these graphs, the higher the dot or bar, the higher the productivity or speed.

Within each project category, the results of other analyses are shown where statistically possible and of interest, for example whether the median[1] PDR varies by language level (3GL, 4GL), specific programming language, etc. The report contains almost no benchmark data on how PDR varies with technical platform, as do standard ISGSG benchmark reports, for two reasons. First, there was insufficient data available to make some of ISBSG's usual main-frame vs mid-range vs PC platform comparisons. For example, projects that used COBOL were almost always developed to execute on main-frames. Second, many projects produced software to be distributed across multiple layers or platforms. The analysis of data from these projects is complex and may be tackled at a later date.

Ideally, the overall performance of any single project should really be determined by examining three parameters:

- Productivity (or PDR)
- Speed (size/elapsed time) and
- Quality (measured via „defect density", i.e. defects/size)

(Actual versus estimated project effort and time should also be considered, though these data concern other aspects of performance).

It is important to consider all three parameters because they may be traded and the trade-offs can be very significant. For example a project may be delivered faster than „norm" by adding more staff to the team, resulting in lower productivity (higher PDR). The delivered software from a „high-speed" project may also be of lower quality (= higher defect density) due to less time spent on defect removal and testing. When examining the PDR for any one project, it is therefore important to compare not only its PDR against the median for its category (i.e. the benchmark) but also to consider the project speed and quality in order to see if the project had experienced a significant trade-off of effort with time and/or quality.

Unfortunately, only a small proportion of projects have submitted any data about defects discovered during the first month of live running. It may be that the vast majority of projects actually had zero defects, but this seems unlikely – we guess that data-submitting organizations either do not record defects or have not bothered to report their defect counts. So in this report it has not been possible to work out if quality has been traded for productivity or speed. However, most projects recorded their duration as well as effort, so in this report it has been possible to establish benchmarks for both productivity (PDR) and speed. This

[1] For most benchmarks, „median" figures (and percentiles) are reported by the ISBSG rather than „average" figures, since an average can be seriously affected by outlier projects. The impact of outliers is clearly undesirable for the purpose of establishing benchmarks. The median of a distribution of data values is the point in the distribution where 50% of the projects have higher values and 50% have lower values. The median is thus the most probable value for the distribution.

should enable users of these data to establish whether or not an individual project has traded effort for time.

The quality of the data submitted appears to be generally good, though it should be mentioned that ISBSG has not means of checking the accuracy of submitted size and effort data. Only projects with „A" and „B" quality data, as judged by ISBSG, were analysed. However, a few probable anomalies were discovered and the data for such projects was not used. This elimination of outliers has to be done with care. On the one hand outlier projects may be particularly interesting to study. On the other hand, outlier project data can sometimes distort benchmark results, which is clearly undesirable when the benchmarks must be used for estimating.

This report has been distributed free to all organizations that submitted COSMIC-measured project data to the ISBSG up to May 1st 2009. Any feedback on the report from data-submitting organizations will be welcome, to ISBSG directly at: admin@isbsg.org (so as to maintain anonymity of the submitting organization).

The detailed raw data for the ISBSG projects analysed in this report are available from the ISBSG. The COSMIC sized projects are available in the ISBSG COSMIC data Release 1. A licence to use this data can be purchased from www.isbsg.org/products.

2. Real-time Application, Software Component and 'Miscellaneous' Software Projects

2.1 General

2.1.1 Overview of the data analysis

The 61 projects comprised 49 new development projects and 12 enhancement projects

The reader is warned that as there are relatively few projects, the statistical uncertainty on some of the benchmark findings is high. However, in general the project data showed reasonable consistency and the results quoted here are ones that the author judges as reasonable and that should be of interest.

The first step of the analysis was to sort the project data into separate sub-sets. Given the few projects, the following sub-sets were distinguished with sufficient numbers to justify an analysis:

- New development projects – real-time application software (19); software components (22); „miscellaneous" software (8)
- Enhancement projects – real-time software only (12)

The first categorization into three types of new development projects involved some judgement in certain cases since the ISBSG categories of „application type" have not yet been adapted enough to clearly distinguish all the different types of software that may be sized by the COSMIC method. This will be explained further in the relevant sections.

2.1.2 Data quality screening

Within each sub-set, the project effort and elapsed time data were examined for credibility or, if reported, for consistency with the reported team size.

For this test, the reported „actual effort" (in work-hours – see Appendix A) was converted to „work-months", by dividing by 120, a typical industry-average for effective work-hours per work-month. Next, dividing work-months by the reported project duration (in elapsed months) gives a „calculated average team-size" for the project., which can be checked for plausibility. As a result of this test, the data for one real-time application new development project was eliminated[2].

Only 15 of the 60 projects submitted counts of defects recorded in the first month of live running and most of these were in the „software components" category. Eight (8) projects reported one „major" defect (on a scale that has minor, major and severe categories), five reported zero defects and three reported one or more minor defects. It may be that real-time application and infrastructure projects, by the nature of their use, are constrained to deliver defect-free software, so where no defect-count was recorded, this was because zero defects were observed, but this is conjecture.

[2] The eliminated project had by far the highest productivity, i.e. the lowest PDR, in its sub-set. Its calculated average team size was 0.35, implying that only one person was working on the project for 35% of the time, but it reported 1.6 as its average team size. Either the reported effort or duration, or both, must be incorrect.)

2.2 Real-time Application Software: New Development Projects

2.2.1 Project demographics

The 18 projects that were assumed to be included in this category recorded their „application type" as:

- Air traffic management (2)
- Central command/control of sensors (1)
- Message switch/cell phone (2)
- Network management (3)
- Process control (1)
- Simulator (1)
- Software for machine control (process control) (1)

or they recorded their „Organization type" as

- Telecommunications (5)
- Communications (1)
- Computers and software (1)

Other demographic data are as follows (the numbers do not always add up to 18 due to unrecorded data)

Architecture: Stand-alone (8); client-server (2)
Development platform: Mid-range (8); PC (5); Multi-platform (2); Hand-held (1)
Language type: 3GL (14); 4GL (1).

The projects ranged in size from 8 to 810 CFP. The effort needed ranged from X to over 19,000 work-hours (i.e. up to 13 work-years); project duration varied from Y to Z2 elapsed months.

2.2.2 Project productivity (PDR)

Figure 1 below shows the Effort (in work-hours) versus the Size (in CFP) for these 18 projects. They appear to be a relatively homogenous set.

The benchmark PDR figures (WH/CFP) from this set (where P = „percentile") are:

N	Min	P10	P25	**Median**	P75	P90	Max
18							

The PDR of the projects in this category was not obviously influenced by any of the architecture, language-type, etc., parameters. For instance the only project that used a 4GL programming language, had a PDR close to the median.

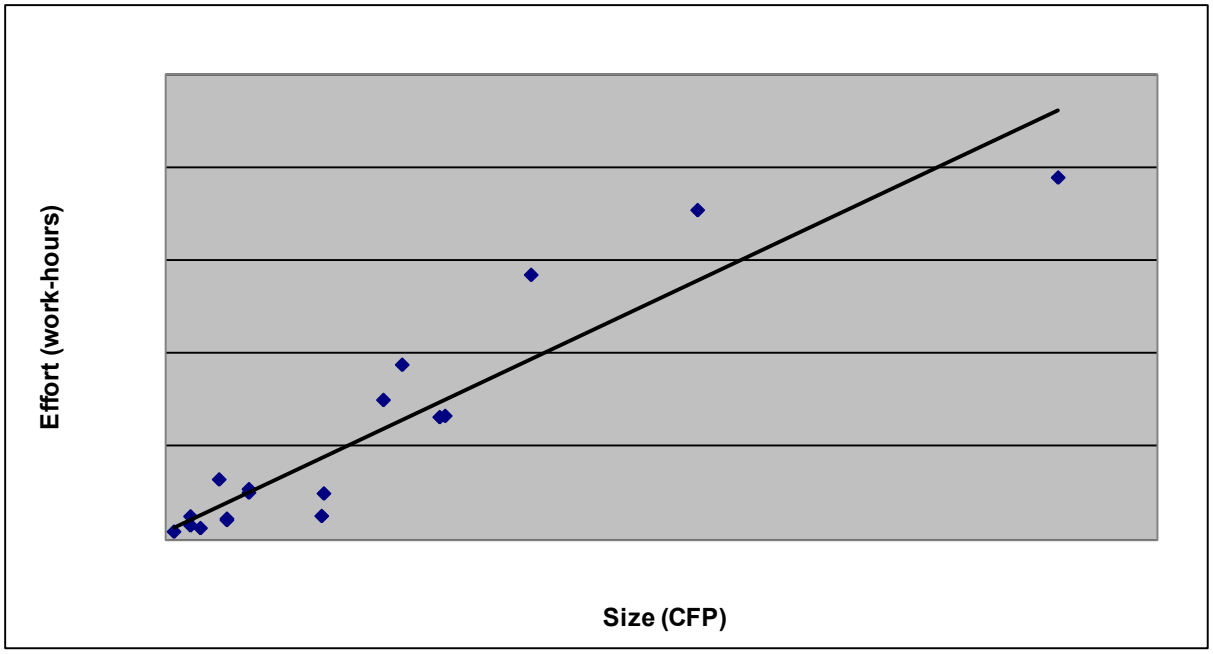


Figure 1 Real-time new development projects: Effort vs Size

2.2.3 Project speed

The software size delivered versus project elapsed time was investigated for these same projects. An initial plot of time versus size indicated three serious outliers. One of these was a project with a very low speed that had lasted 54 months of which 20 had been inactive. The other two projects were found to have „calculated average team sizes" of 14 and 29. Both figures are implausible and probably result from under-recording of the elapsed time, though the effort data seem to be OK.

After eliminating these three projects, plus one other project that did not report its duration, Figure 2 shows the duration, measured in elapsed months, versus size for the remaining 14 projects.

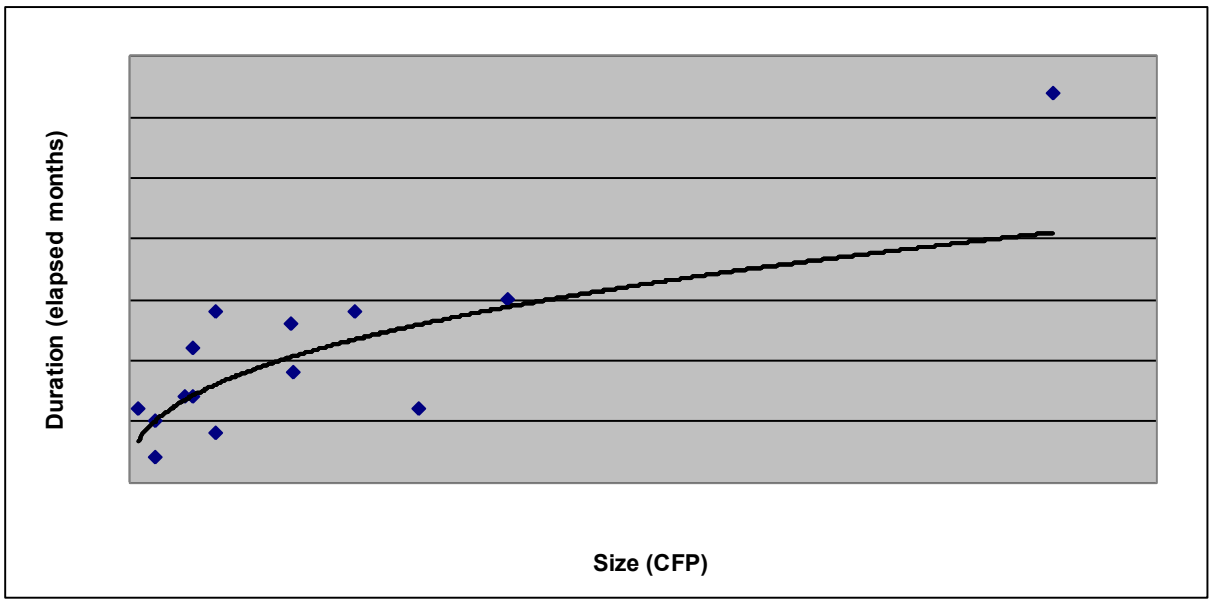


Figure 2 Real-time new development projects: Duration vs Size

Project speed from this power curve is then given, in units of CFP per elapsed month, by the formula:

Speed = A x (Size) ^ B

2.3 Software Components: New Development Projects

2.3.1 Project demographics

Two sets of data fit into this category. They are:

- 15 projects developed in an Engineering R&D organization that delivered software described as „Glue software" from a „MIS Linguistic software system", The software was developed for a client-server architecture using Oracle technology on a PC platform
- 7 projects developed in an insurance company that delivered re-usable components for an operating system or utilities, using Java, on a PC platform

These were small projects. They ranged in size from 8 to 470 CFP, needing from X to Y work-hours (i.e. up to about 5 work-months); project duration varied from 1 day to Z elapsed months.

2.3.2 Project productivity (PDR)

Figure 3 shows the Effort (in work-hours) versus the Size (in CFP) for these 22 projects. It appears to be reasonable to treat these data as belonging to one set.

The benchmark PDR figures (WH/CFP) from this set (where P = „percentile") are:

N	Min	P10	P25	**Median**	P75	P90	Max
22							

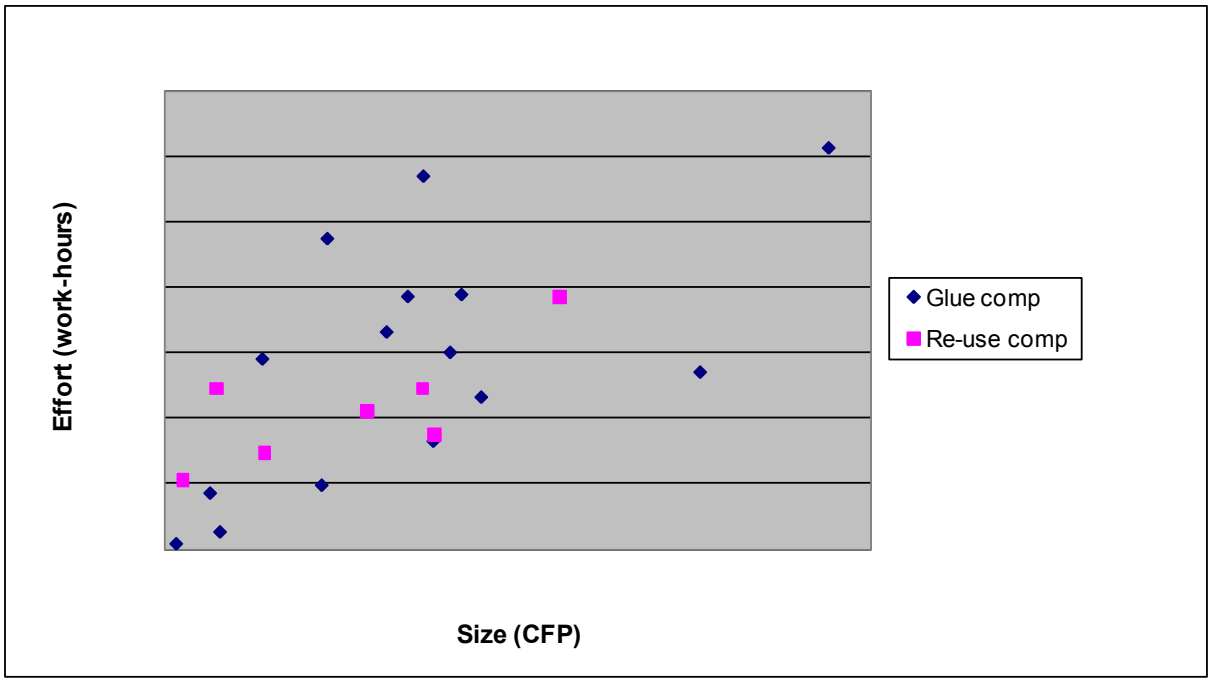


Figure 3 Software component new development projects: Effort vs Size

2.3.3 Project speed

Only the „glue software" component projects reported credible project durations. Figure 4 shows the elapsed months versus size for these 15 projects.

These „glue software" projects are all „small" projects, requiring on average 2.4 work-months of effort and 1.3 elapsed months duration. All but four of the projects appear to have had a team size of two persons. Not surprisingly therefore, there is quite a scatter in productivity and speed of the projects.

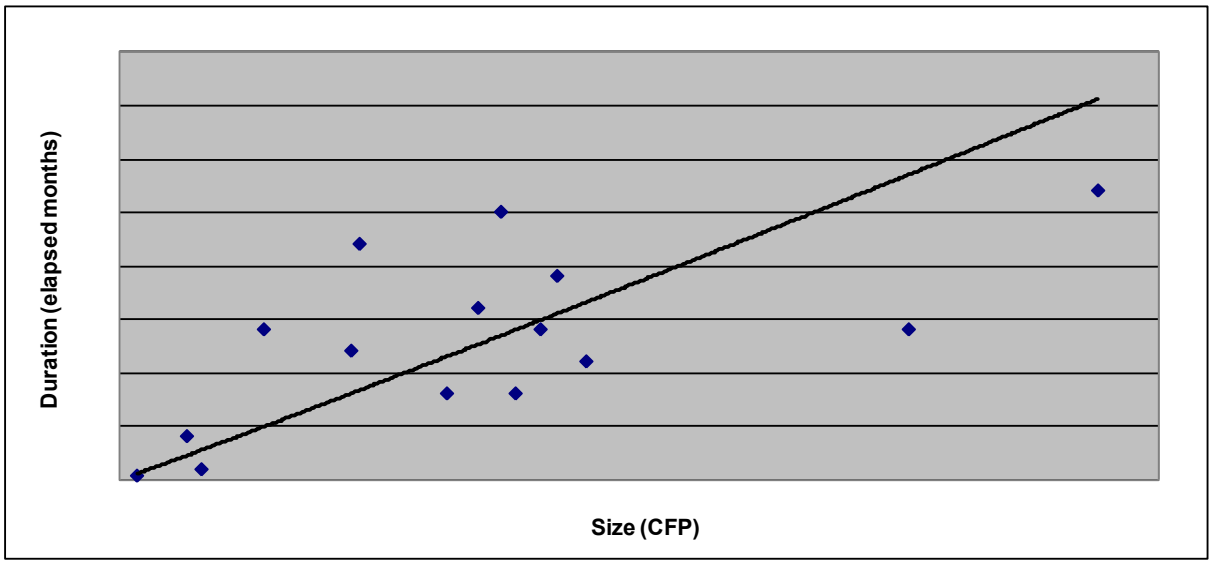


Figure 4 'Glue' component new development projects: Duration vs Size

The median speed of these „glue" software component projects in units of CFP/elapsed month is about 10 times greater than the median speed of the real-time application software projects for the same size of software delivered. Similarly, the median PDR of the „glue" software component projects at X Work-hours per CFP is almost 20 times lower (i.e. higher productivity) than the median PDR of the real-time application software projects.

These results illustrate why it is imperative not to mix performance results of projects developing pieces of software at different levels of decomposition.

It is imperative not to mix performance results of projects developing pieces of software at different levels of decomposition.

2.4 Miscellaneous Software: New Development Projects

The data supplied by 8 projects did not allow them to be included in the same two categories described above, nor as business application projects.

Six of these projects have the following characteristics (in descending order of PDR):

Application description	Hardware	Language	Size-CFP	Effort-WH	Elaps Mth	PDR	Speed	Av team size
Register of events - OS utility	Multi-tier	3GL/Java	84					
Graphical modelling	Cl-Svr PC	4GL.Net	1511					
Graphics publishing tools or system	Cl-Svr PC	3GL/Java	79					
Fault tolerance	SA PC	4GL/Vis Basic	35					
Software for machine control	Cl-Svr PC	4GL.Net	1384					
Software for machine control	Cl-Svr PC	4GL.Net	588					

(Note: „Cl-Svr" = client-server; „SA" = stand-alone.)

The first three of these six projects have similar performance to the real-time application projects discussed in section 2.2.

The last two projects have performance characteristics that suggest they may fit in the category of „software components" discussed in section 2.3, though the CFP sizes are larger than the software produced by other projects in that category.

The „fault tolerance" software may also fit in this „software components" category, although the reported elapsed time for this project is not credible.

The remaining two of the eight miscellaneous projects delivered mathematically-intensive software. Their characteristics are shown below.

Application description	Hardware	Language	Size-CFP	Effort-WH	Elaps Mth	PDR	Speed	Av team size
Algorithm + DB	SA PC	3GL Ada	484					
Geographic or spatial info system;	Cl-Svr	3GL C++	106					

These two projects have a PDR significantly higher (lower productivity) than any of the real-time application software projects and a speed that is about half that of the real-time projects of the same size. Note that the low productivity probably results from the fact that the COSMIC FSM method is currently unable to properly account for heavy mathematical processing in its measure of functional size – as is also true for all other standard FSM methods.

2.5 Real-time Application Software: Enhancement Projects

2.5.1 Project demographics

The data on 12 enhancement projects reported their „application type" as:

- Network management (2)
- Sensor control & presentation (1)
- Software for machine control (2)

or their „organization type" as:

- Telecommunications (7)

The projects ranged in size from 23 to 624 CFP. The effort needed ranged from X to over 25,000 work-hours (i.e. up to 17 work-years); project duration varied from Y to Z elapsed months.

2.5.2 Project productivity (PDR)

Figure 5 shows the effort in work-hours versus size for these 12 enhancement projects.

Examining these data in detail, six of the seven telecommunications enhancement projects used a 4GL language, the seventh used a 3GL language. All executed on either a Multi-platform or Mid-range hardware platform; it seems likely the data all come from one organization. They form a coherent set with a median PDR of X WH/CFP.

In addition, two other projects delivered 624 CFP (a far larger size than any other project) and 72 CFP, both with a PDR of Z WH/CFP – the same as the median for the seven telecommunication/4GL projects. Their programming languages were Ada and TNSDL, respectively.

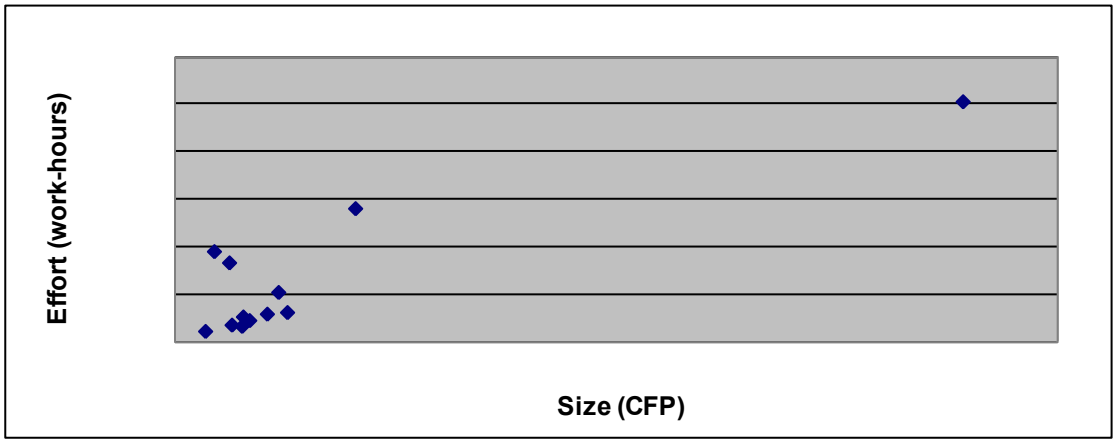


Figure 5 Real-time enhancement projects: Effort vs Size

The three projects that had a very much lower productivity were as follows:

- Two projects that enhanced „software for machine control" delivered 42 and 30 CFP with a very high PDR of X and Y WH/CFP respectively. They were programmed in C, on a mid-range platform.
- Another project delivered 142 CFP with a high PDR of Z WH/CFP. This was also programmed in C, on a stand-alone PC.

Other than noting that using the C language seems to result in much lower productivity than the other 3GL languages, the statistics are not such that it seems valid to separate these projects into sub-groups. The PDR figures also seem to be independent of the hardware platform.

For these 12 projects, the PDR figures are as in the table below.

N	Min	P10	P25	**Median**	P75	P90	Max
12							

Excluding the three low productivity projects that used the C language, the PDR figures would be:

N	Min	P10	P25	**Median**	P75	P90	Max
9							

2.5.3 Project speed

Figure 6 shows the duration measured in elapsed months versus size for the 12 real-time enhancement projects.

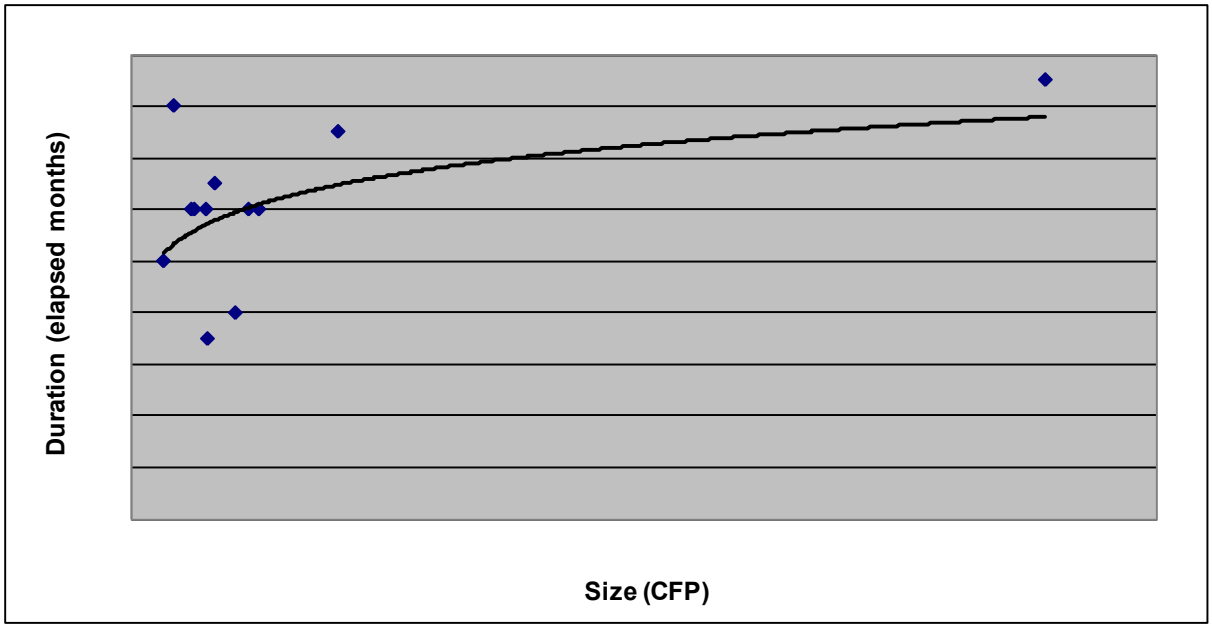


Figure 6 Real time enhancement projects: Duration vs Size

Two of the three projects that had a high PDR (low productivity) also have a low speed.

3. Business Application Software Projects

3.1 General

3.1.1 Overview of the data analysis

Data on 272 business application projects were reported of which 163 were from the banking industry. Other large industry categories represented were:

- Government and Public Administration (27)
- Insurance (24)
- Engineering (14)
- Medical & Healthcare (6)
- Retail & Wholesale trade (5)

The 272 projects comprised 129 new development projects, 137 enhancement projects and 6 re-development projects. These three sub-sets were analysed separately.

The new development and enhancement projects were further analysed by language level and type, and to investigate how performance varies with the size of the software delivered.

3.1.2 Data quality screening

Within each sub-set, the project effort and elapsed time data were examined for credibility or, if reported, for consistency with the reported team size.

For this test, the reported „actual effort" (in work-hours – see Appendix A) was converted to „work-months", by dividing by 120, a typical industry-average for effective work-hours per work-month. Next, dividing work-months by the reported project duration (in elapsed months) gives a „calculated average team-size" for the project.

As a result of this test, the data for two new development projects were eliminated as not credible, and two projects did not report any work-hours, so their data could not be used.

Only 29 of the 272 projects reported counts of defects recorded in the first month of live running. Another 24 projects reported that they had found zero defects in the first month of live running. With very few exceptions, the defects reported were mostly „minor" defects, with a few „major" defects. These data were therefore not analysed further.

3.2 New Development Projects

3.2.1 Project demographics

The 125 new development projects with usable data ranged in size from 10 to 1670 CFP.

Effort ranged from under U to over 46,000 work-hours (i.e. up to approximately 32 work-years), with a median of V work-hours (i.e. approximately W work-years). Duration ranged from 1 to X months, with a median of Y months.

.

All projects involved bespoke developments, i.e. no projects involved any package customisation.

3.2.2 Project productivity (PDR)

Figure 7 shows the effort (in work-hours) versus the size (in CFP) for these 125 projects. This graph distinguishes projects that used a 3GL or a 4GL language.

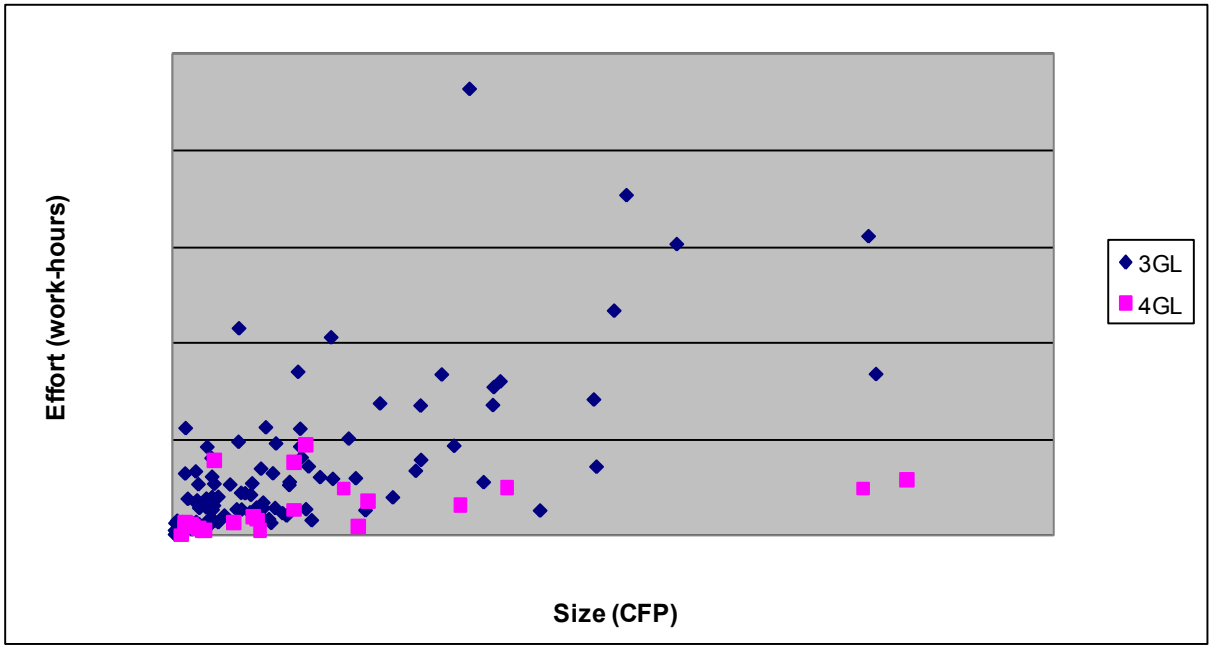


Figure 7 Business application new development projects: Effort vs Size

The benchmark PDR figures (WH/CFP) from this set (where P = „percentile") are:

Language Level	N	Min	P10	P25	**Median**	P75	P90	Max
3GL	100							
4GL	25							

Projects developed using a 4GL programming language are roughly 2.5 times as productive as those using a 3GL language.

Breaking down the data for the 3GL projects by language type and ignoring four projects that used miscellaneous languages, we obtain the following PDR benchmark data for the most commonly used languages.

Language Type	N	Min	P10	P25	**Median**	P75	P90	Max
C# & C#.Net	15							
COBOL	46							
Java	31							
Visual Basic	4		-	-		-	-	

(For possible refinements to these median figures, see section 3.2.5.)

The distribution of the 3GL projects over hardware platforms is as below.

	Main-frame	**Multi-platform**	**PC**	**Unknown**	**Total**

C# & C#.Net	-	14	-	1	15
COBOL	46	-	-	-	46
Java	13	11	7	-	31
Visual Basic	-	2	2	-	4

Given this mapping of programming language and hardware platform for these projects, it is only of interest to compare the median PDR of the 31 Java projects across the three platforms. The median figures expressed in Work-hours/CFP are:

Mainframe – X; Multi-platform – Y; PC – Z

The 25 projects that used a 4GL programming language used eight principal language types (most commonly ASP, .Net, and Oracle) on all three main platforms. The numbers are too small to draw any conclusions about any variation of PDR with language type or platform.

Project PDR also varies with the size of the software delivered. Figure 8 below shows how the median project **productivity** for all new development projects varies with delivered software size for various size bands. (Productivity, expressed in CFP/work-month, has been calculated from PDR assuming an average of 120 work-hours per work-month. Productivity is preferred over PDR for this analysis as it is more natural to associate increasing productivity with „improving performance".)

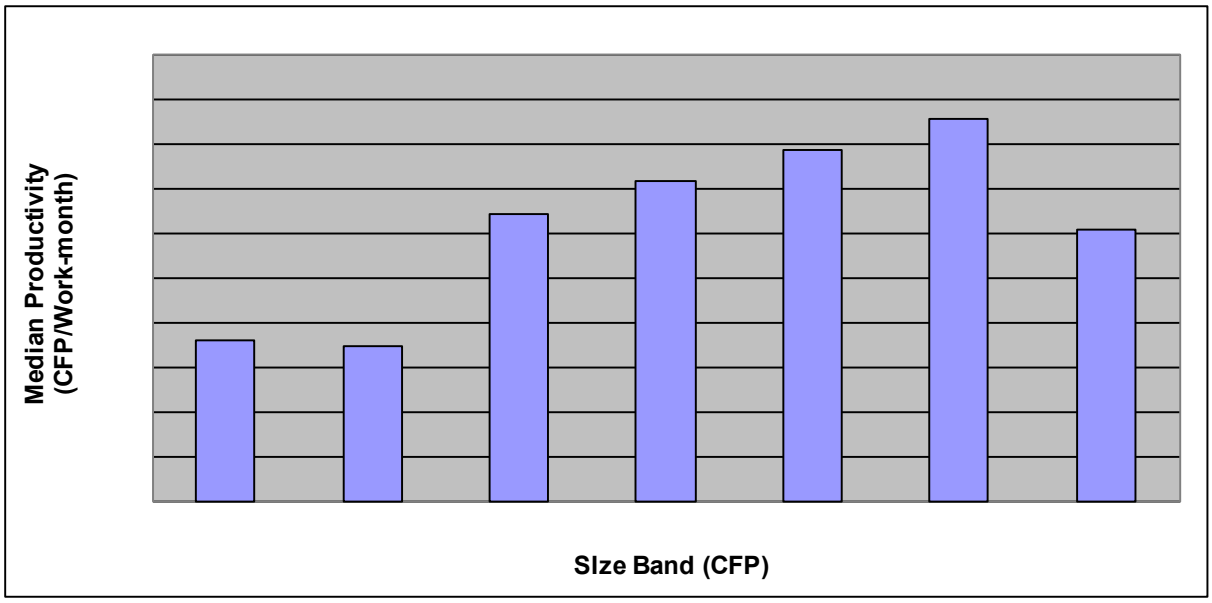


Figure 8 Business application new development projects: Median Productivity per Size Band

The result in Fig. 8 is economically important in showing an economy of scale for new development projects of size up to around 500 – 1000 CFP.

There is an economy of scale for new development projects with size of software delivered up to the range of 500 – 1000 CFP.

However, digging deeper into the data in Figure 8, limiting to new development projects that used a 3GL programming language shows that the spread of productivity, as measured by the percentiles, varies significantly with size band, as shown in Figure 9.

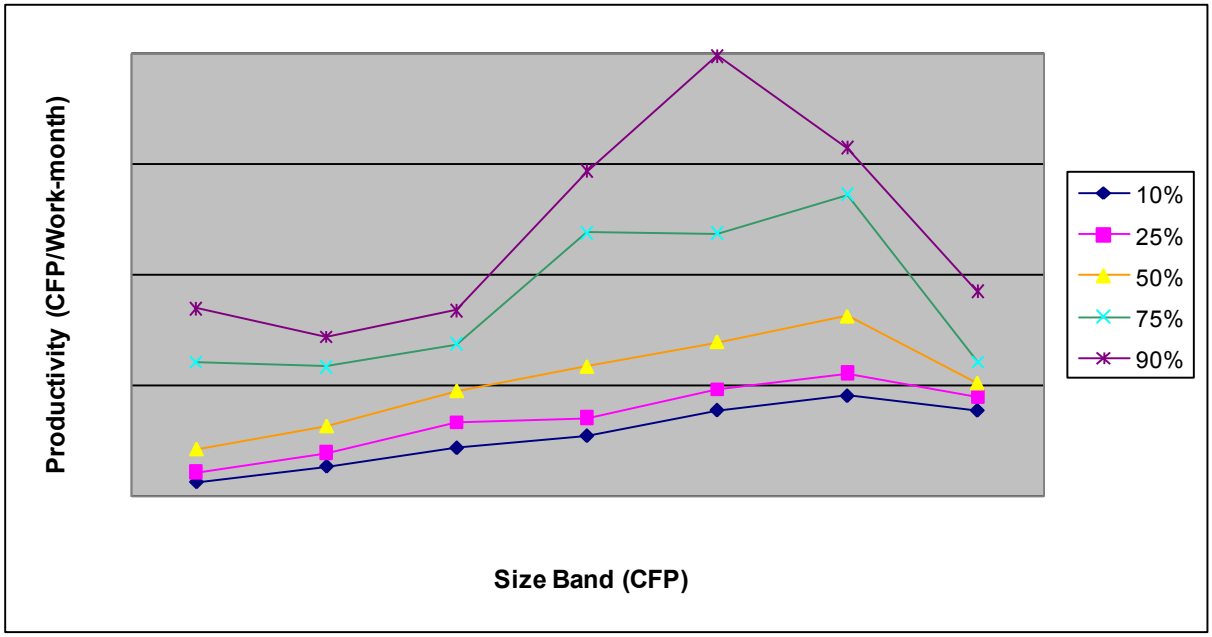


Figure 9 Business application new development 3GL projects: Percentiles of Productivity per Size Band

Figure 9 indicates that not only does productivity increase with size, up to around 1000 CFP, but the spread of productivity widens sharply. This must be due to other factors than size, perhaps risk. This finding is important for estimating accuracy.

Not only does productivity increase with size, up to around 1000 CFP, the spread of productivity widens sharply. This must be due to other factors than size, perhaps risk. This is important for estimating accuracy.

Figures 8 and 9 also indicate that productivity declines above 1000 CFP, though the number of projects is small in this band.

(In order to help judge the significance of these two results, the number of projects included in each size band is shown below.)

	Number of projects by Size Band (CFP)						
Size Band ->							1000+
All projects (Fig. 8)	18	30	17	23	14	17	7
3GL projects (Fig. 9)	13	26	14	18	10	14	5

3.2.3 Project speed

Roughly one-third of the new developments of business applications did not report their project duration. Figure 10 shows the duration, measured in elapsed months, versus size for 82 new development projects, distinguishing those that used a 3GL programming language from those that used a 4GL language. (A 3GL project of 65 months duration and a 4GL project of 35 months duration have been removed as outliers.)

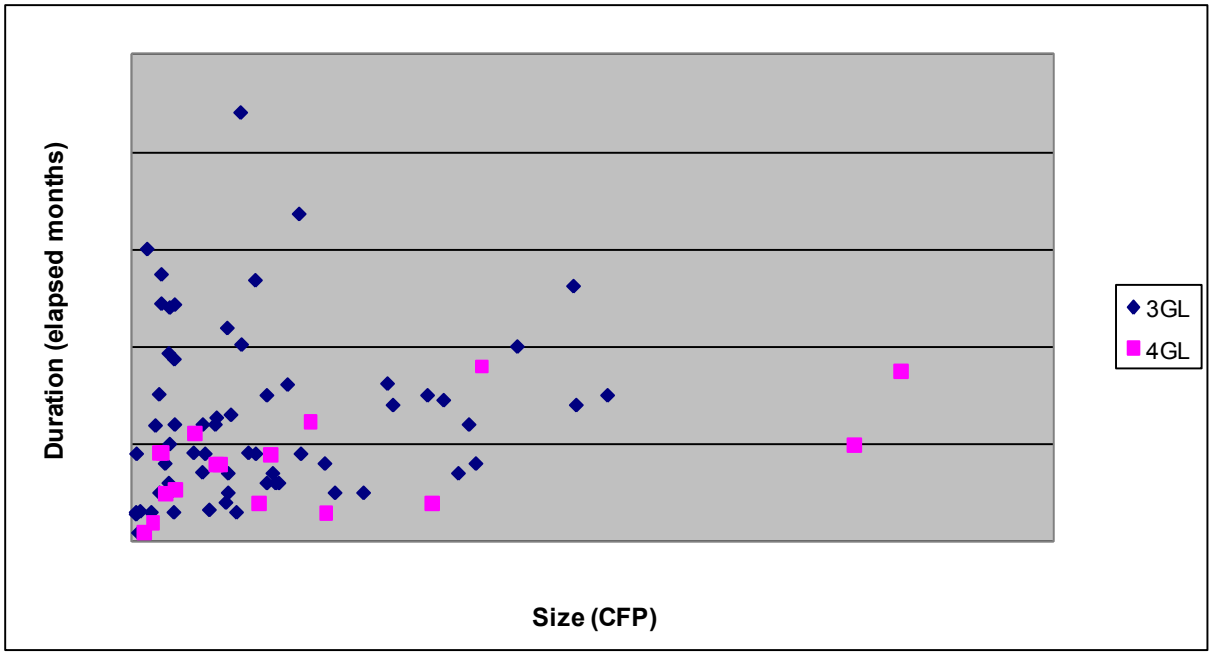


Figure 10 Business application new development projects: Duration vs Size

Figure 11 shows the median project **speed** (Size / elapsed months) for the 3GL and 4GL projects in the same size bands as those used for the median productivity in Figure 8. This chart shows that project speed increases sharply with size of the software delivered, i.e. there is an important economy of speed as well as of scale.

Project speed increases sharply with size of the software delivered, i.e. there is an important economy of speed as well as of scale.

The chart also indicates that projects that use a 4GL language can be developed roughly twice as fast, for a given size, as those that use a 3GL language.

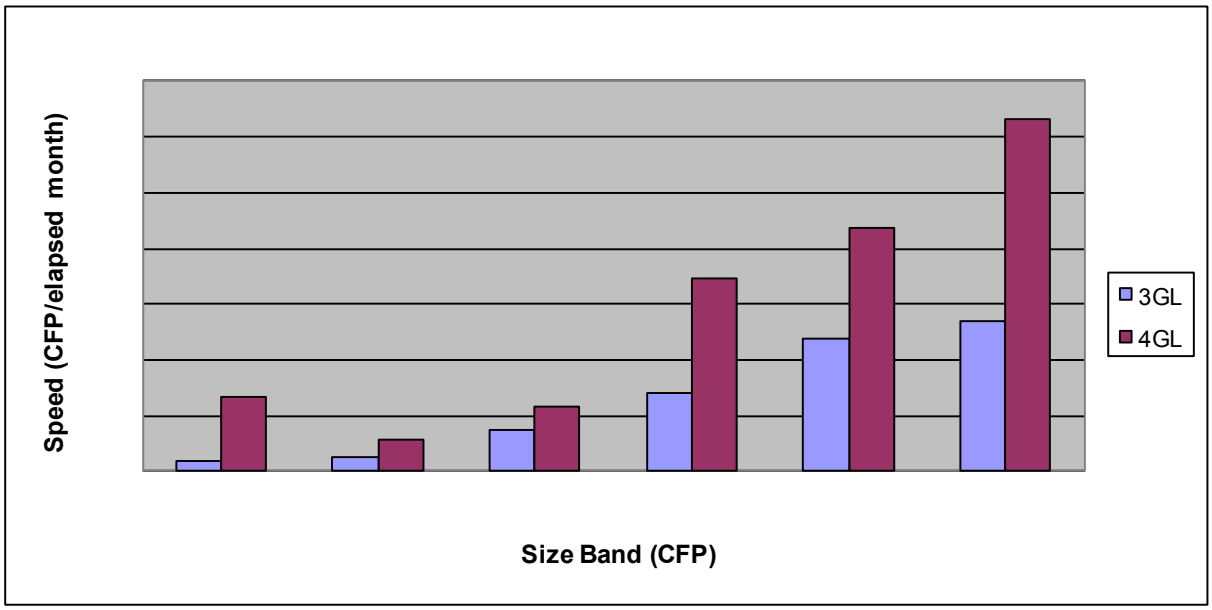


Figure 11 Business application new development projects: Median Speed vs Size Band

Projects that use a 4GL language can be developed roughly twice as fast, for a given size, as those that use a 3GL language, as well as 2.5 times as productively.

(In order to help judge the significance of this result, the number of projects included in each size band is shown below.)

Size Band (CFP) ->	Number of projects by Size Band					
3GL projects	7	15	7	13	8	12
4GL projects	2	4	3	1	4	4

Fitting a power curve to the median speed (CFP/elapsed month) versus the median size (CFP) for each band gives the following formulae, which should be useful for estimating.

3GL projects Speed = A x (Size)^B (or Months = C x (Size)^D)
4GL projects Speed = E x (Size)^F (or Months = G x (Size)^H)

However, digging deeper into the data of Figures 10 and 11 we find that, for a given size, project duration varies significantly with the choice of 3GL language. Figure 12 shows the duration, measured in elapsed months, versus size for 54 projects, distinguishing the three 3GL languages that were mainly used.

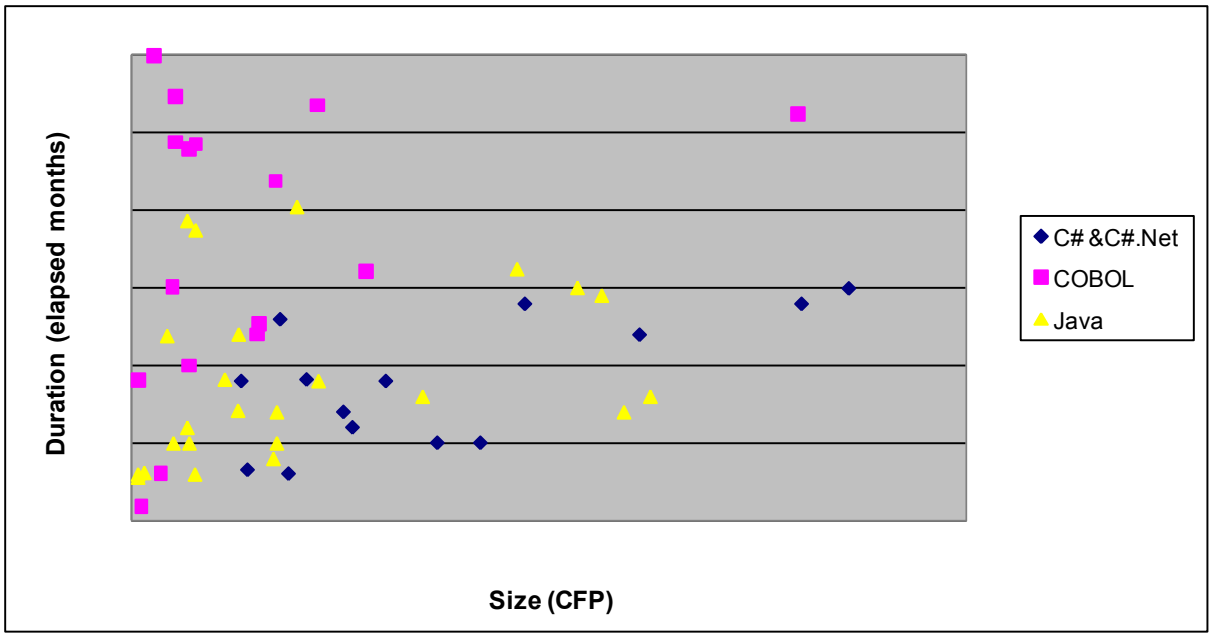


Figure 12 Business application new development projects : Duration versus Size

This chart shows that for a given size, COBOL projects tend to take longer than Java and C# or C#.Net projects. Further, the duration of COBOL projects tends to be more variable, especially at lower sizes than Java projects which in turn are similarly more variable in duration than C# and C#.Net projects.

Due to the very large scatter of elapsed times for these projects, especially for smaller software sizes, fitting any power curve to the data for the different languages would not be statistically very meaningful.

3.2.4 Development of batch versus on-line software

A parameter that ISBSG data collection questionnaires do not capture is whether software to be developed should execute in batch or on-line mode, or a mixture of modes. Some data received privately by the author from a company in the financial services industry (referred to as „Finco" for this report) concerned 26 projects, of which 23 are new developments of software programmed using COBOL.

The data included the effort on Design, Build and Test activities, and on whether the software should operate in batch, on-line or mixed mode. Because the effort data does not cover the activities of Plan, Specify and Implement, the PDR figures quoted in this section should not be compared in absolute terms with those quoted above for COBOL new development projects. It is the relative PDR of projects that produced batch versus on-line, versus mixed mode software that is of interest.

Further, due to the small sample sizes the data were checked, and in part corrected, for the fact that the three groups had different average size of software developed. (The results of section 3.2.2 show that productivity of new development projects varies significantly with software size produced; these results were used to provide the correction.)

The PDR figures for the 23 new development projects are as follows.

Software execution mode	**No. of projects**	Median PDR (WH/CFP)	Median size of projects (CFP)	PDR corrected for size differences	PDR relative to On-line PDR
Batch	15				
On-line	4				
Mixed	4				

The table shows that projects that must develop batch software require X times as much effort per CFP (or are X times less productive) than projects that must develop on-line software. Projects that must develop software operating in mixed mode are even less productive.

The explanation for these results might be that:

- the technology for developing on-line software delivers functionality, e.g. GUI interfaces, particularly easily compared with that used to develop batch software
- teams developing batch software systems must often negotiate interfaces with other systems, the effort which lowers productivity.

Although the statistics behind this result are very limited, the finding that projects delivering batch software are significantly less productive than projects delivering on-line software is consistent with that obtained when using the MkII FPA Method to measure software sizes, with much larger numbers of projects (see section 6.2).

3.2.5 Some second-order effects on the benchmarks

We have seen that productivity improves with size of the delivered software (see Figure 8). It is therefore important to check whether the projects whose data were used for the benchmark PDR figures given in section 3.2.2 for the different programming languages were of the same average size. Differences in average size of the projects in each programming language group could have distorted the relative median PDR's for the different languages.

The table below shows that the (rounded) average size of software delivered by the projects for each of the four 3GL languages and for the projects that used a 4GL language does vary across the groups.

	C#	COBOL	Java	VB	4GL
Av size of delivered software (CFP)	450	250	250	450	350
No. of projects	15	46	31	4	25

If we then use the data of Figure 8, to suppose that all five groups of projects had delivered software of the same average size as the COBOL and Java projects (250 CFP), then the median PDR figures in the table in section 3.2.2 for the C# and VB projects would be up to about 10% higher (to X and Y WH/CFP, respectively), and the PDR of the 4GL projects would be a few percent higher (but remain at Z WH/CFP, rounded).

Alternatively, data on the varying mix of programming language used with projects in the different size bands could be used to correct the productivity trends of Figure 8. Such a correction would result in a slightly smoother increase in productivity with size up to 1000 CFP.

To find the correct balance between the influence of programming language mix on the productivity / size relationship, versus the influence of varying size mix on programming language productivity benchmarks would require a much more sophisticated statistical analysis applied to a very much larger dataset. We conclude from this brief analysis that these second-order effects are not very important for the analyses of this report.

Similar effects of varying programming language mix on project speed versus size of software delivered would also be expected.

3.3 Enhancement Projects

3.3.1 Project demographics

Data was reported on 123 projects to enhance business application software[3]. With the exception of two very large projects, they ranged in size from a project delivering 3 CFP, taking 24 work-hours effort, up to projects delivering X CFP and taking over 21,000 work-hours (i.e. approximately 15 work-years). The two exceptionally large enhancement projects delivered 1250 and over 2000 CFP, the latter requiring Z work-years of effort.

3.3.2 Project productivity (PDR)

Figure 13 shows the Effort (in work-hours) versus the Size (in CFP) for the 121 enhancement projects, i.e. excluding the two exceptionally large projects.

[3] Data from 14 enhancement projects that used the C++ programming language had very low PDR figures. It is not clear from the data if these projects were really concerned with enhancing business applications, or possibly infrastructure components. They were therefore excluded from the subsequent analysis

Unlike the business application new development projects, the enhancement projects showed hardly any variation of PDR (work-hours/CFP) with the programming language used.

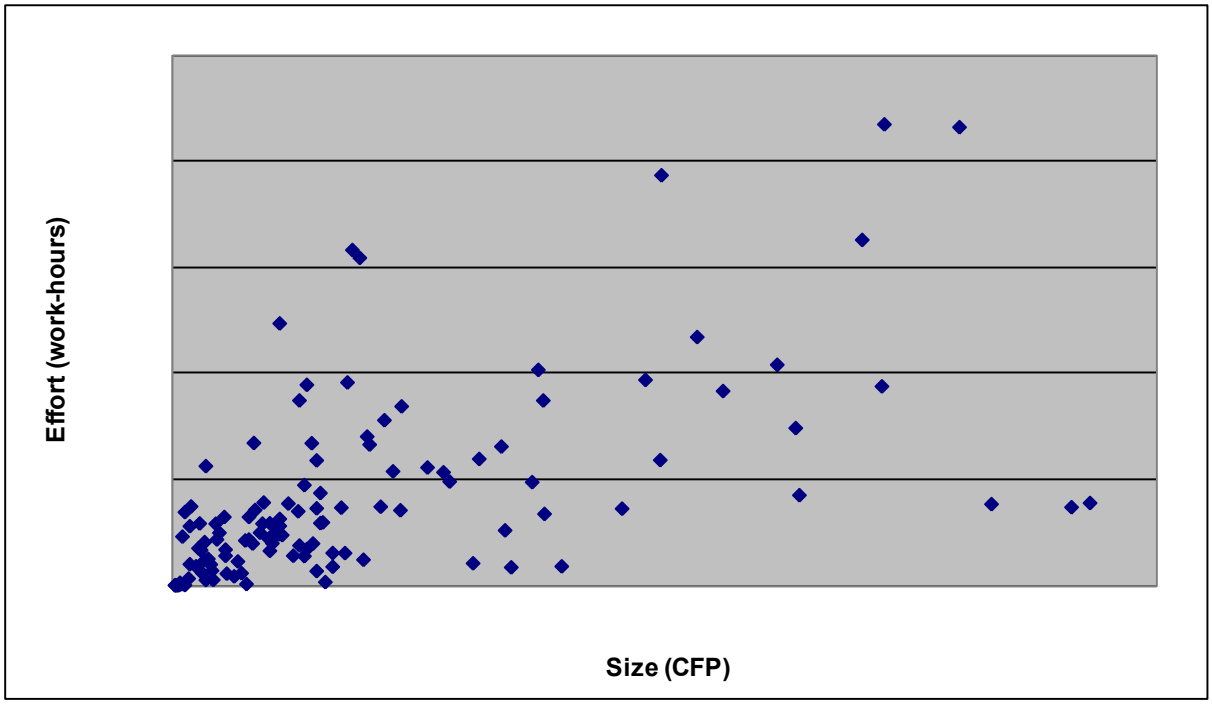


Figure 13 Business application enhancement projects: Effort vs Size

The benchmark PDR figures (WH/CFP) from this set (where P = „percentile") are:

N	Min	P10	P25	**Median**	P75	P90	Max
123							

When the data of 118 projects are analysed by programming language for the principal languages used, the results are as in the following table.

	N	P10	P25	**Median**	P75	P90
COBOL	73					
Java	25					
VB & VC++	9					
4GL	11					

The distribution of these same 118 projects over hardware platforms is as below.

	Main-frame	**Multi-platform**	**PC**	**Unknown**	**Total**
COBOL	65	6	1	1	73
Java	12	-	12	1	25
VB & VC++	1	3	5	-	9
4GL	5	-	6	-	11

The PDR of the projects is quite consistent across the various languages, also considering the differences in hardware platform. Only the projects that used a „Visual" language on a multi- or PC platform have slightly lower PDR (= higher productivity) than the projects that used other 3GL and 4GL languages.

Enhancement project **productivity** also increases with the size of the software delivered.

Figure 14 shows how the median project **productivity** for all 118 enhancement projects varies with delivered software size, using the same size bands as for the new development projects in Figures 8 and 9. (Productivity, expressed in CFP/work-month, has been calculated from PDR assuming an average of 120 work-hours per work-month. Productivity is preferred over PDR for this analysis as it is more natural to associate increasing productivity with „improving performance".)

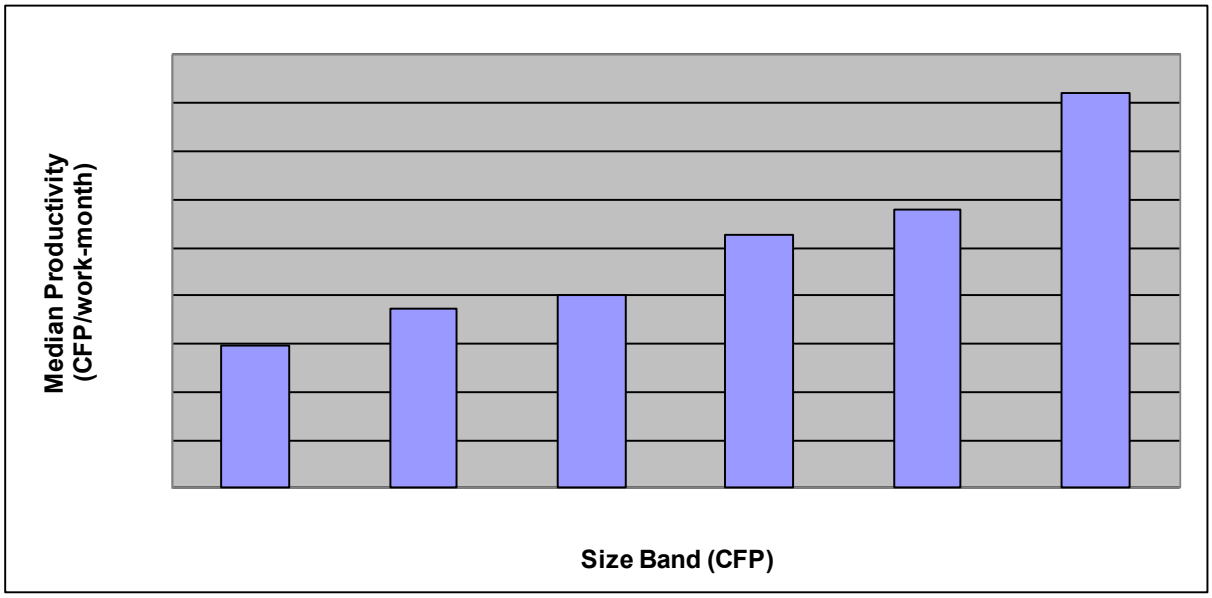


**Figure 14 Business application enhancement projects:
Median Productivity per Size Band**

In order to help judge the significance of these two results, the number of projects included in each size band is shown below.

Size Band (CFP) ->	Number of projects by Size Band					
						500 – 750
All projects (Fig. 14)	36	25	31	10	10	9

Enhancement projects show increasing productivity with size of software delivered up to the range of 500 – 750 CFP, similar to that for new development projects.

Figure 15 shows the percentiles of productivity for enhancement projects, but limited to projects that used a 3GL language, to make the chart comparable with Figure 9 for new development projects.

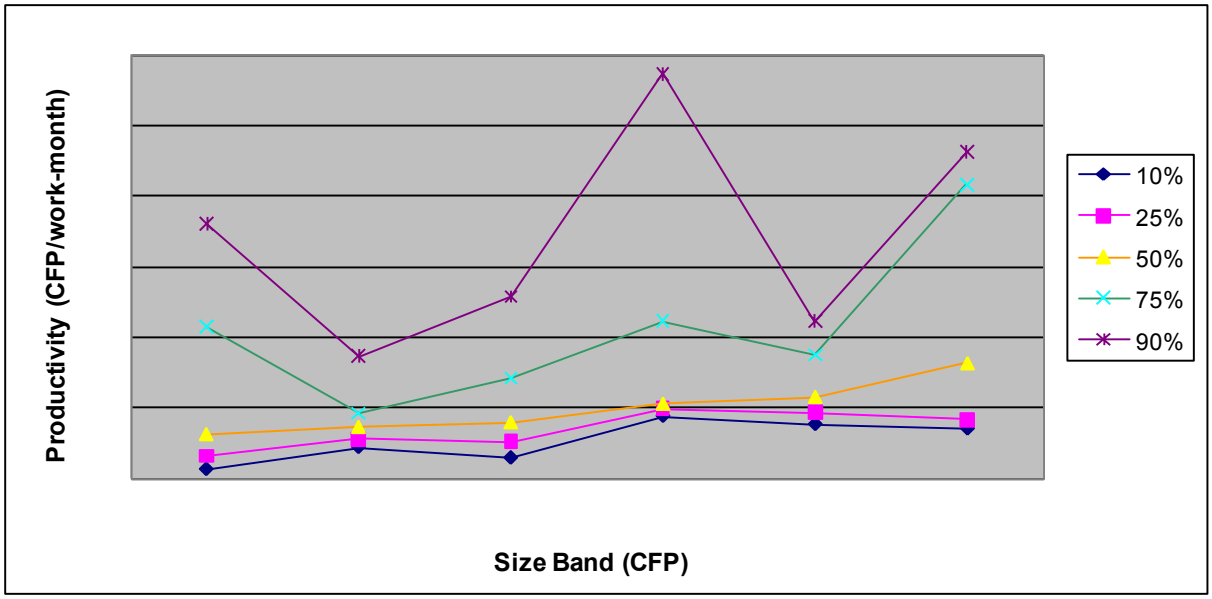


Figure 15 Business application 3GL enhancement projects: Percentiles of Productivity per Size Band

The spread of 3GL enhancement project productivity shown in Figure 15 is noticeably greater than that for 3GL new development projects.

3.3.3 Project speed

Over 40% of the projects to enhance business applications did not report their project duration. Figure 16 shows the duration measured in elapsed months versus size for 63 enhancement projects that used a 3GL programming language and 9 that used a 4GL language. (A 3GL project that produced an enhancement of over 2000 CFP has been removed as an outlier.) There was no significant difference between the speed of 3GL and 4GL enhancement projects.

Unlike development projects, there was no significant difference between the speed of 3GL and 4GL enhancement projects for business applications.

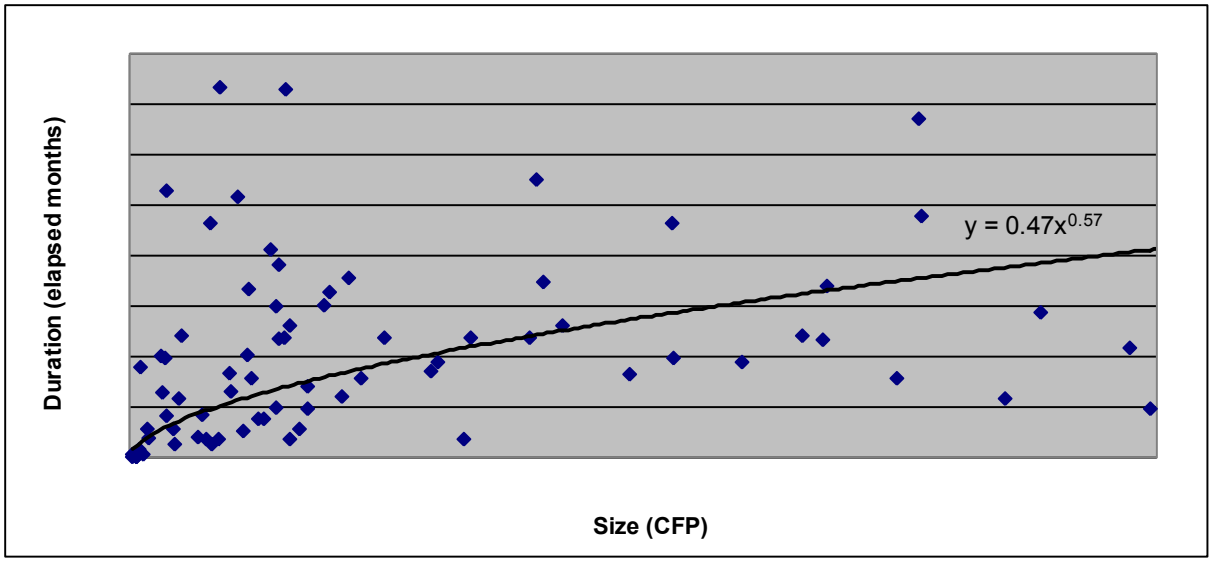


Figure 16 Business application enhancement projects: Duration versus Size

Figure 17 shows the median project **speed** (Size / elapsed months) for the 72 enhancement projects in the same size bands as those used for the median productivity in Figure 14. This chart shows that project speed increases sharply with size of the software delivered, i.e. there is an important economy of speed as well of scale.

Enhancement project speed increases sharply with size of the software delivered, i.e. there is an important economy of speed as well of scale.

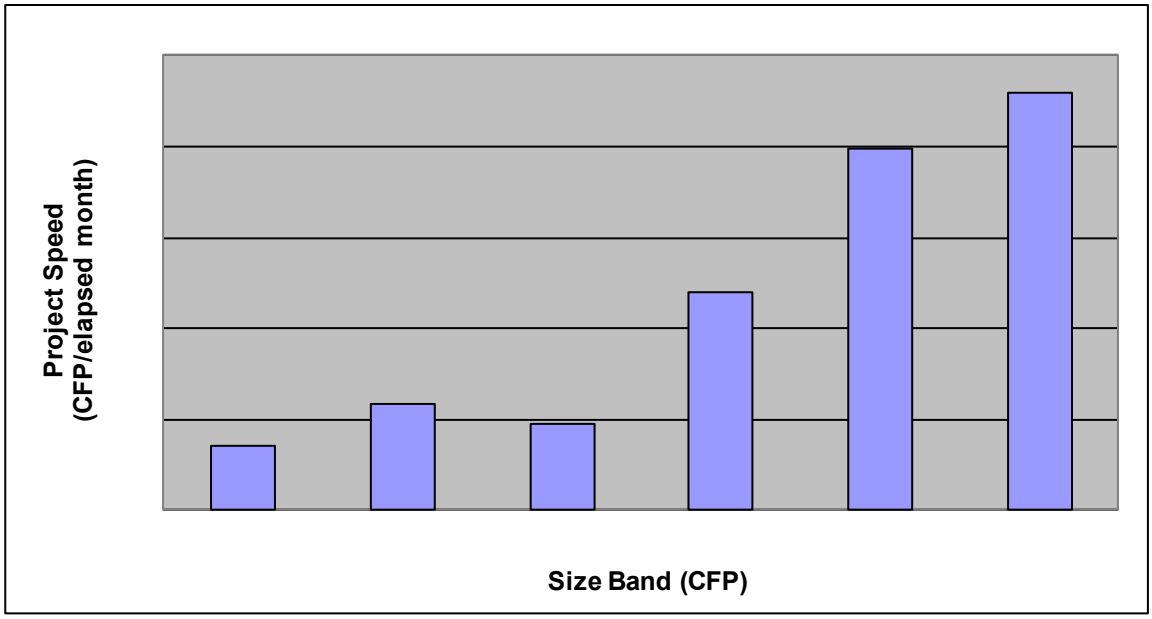


Figure 17 Business application enhancement projects: Median Speed per Size Band

This pattern of increasing speed versus size band for enhancement projects in Figure 17 is very similar to that for new development 3GL projects as in Figure 11. However, enhancement projects appear to be faster than new development 3GL projects up to 100 CFP and slightly slower above that size.

Fitting a power curve to the median speed (CFP/elapsed month) versus the median size (CFP) for each band of these enhancement projects gives the following formulae, which should be useful for estimating.

Enhancement projects: Speed = A x (Size)^B
(or Months = C x (Size)^D)

3.4 Re-development projects

Data were submitted on six business application re-development projects. The projects delivered software that ranged in size from 55 to over 2000 CFP, requiring from X work-hours up to over 20 work-years of effort and taking from Y to Z elapsed months.

Data on five of these projects was reasonably consistent in spite of being concerned with a variety of application types. They were programmed in Java (2), VB (2) or C# to execute on a PC (4) or multi-platform hardware.

For these five projects the PDR figures (Work-hours / CFP) are as follows.

Minimum = X; median = Y; maximum = Z

The best-fit straight line for project speed for these five projects is

Speed = A + B x Size

The one outlier project, which used a 4GL, had a PDR five times higher (i.e. lower productivity) than that of the median of the other five projects, and speed three times slower than would be expected from the equation for the other five projects.

4. Project Effort Distribution

It is important and instructive for estimating to understand the normal distribution of effort over the various activities of a project. As discussed in Appendix A, the ISBSG aims to collect data on six activities, namely Planning, Specify, Design, Build, Test and Implement.

The distribution of effort should in principle be independent of the project design and management method, i.e. whether the project follows a „waterfall" or „agile" process. However, it seems fairly safe to assume that the projects reported on here used a waterfall process in most cases.

As noted in Appendix A, few projects submitted effort data for all six activities; most reported the effort for only 4 or 5 activities. This limits the possibility of producing effort distributions in the detail that is really needed.

4.1 Real-time application new development projects

Only five projects supplied project effort distribution data. The only effort distribution percentages that can be derived from these data are shown in Fig. 18 below.

The effort on planning, specification and design as a percentage of normalised Level 1 effort[4] ranges from 9% to 32% for these five projects. The data for the five projects varied greatly, so the averages in the chart are not very reliable.

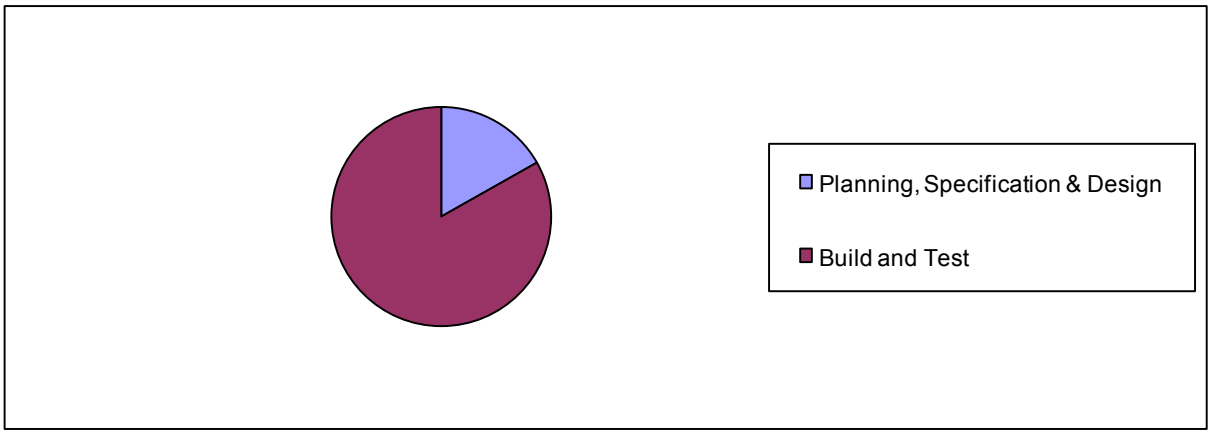


Figure 18 Real-time new development projects

4.2 Business application new development projects

Thirty-five projects supplied data on all of the effort on Specify, Design, Build and Test activities. This enabled the calculation of the percentages of the total effort on each of these four activities. Effort spent on implementation activities was ignored as this varied enormously by project (See Appendix A).

Figure 19 shows these percentages for the 35 projects.

[4] For the definition of Normalised Level 1 effort, see Appendix A

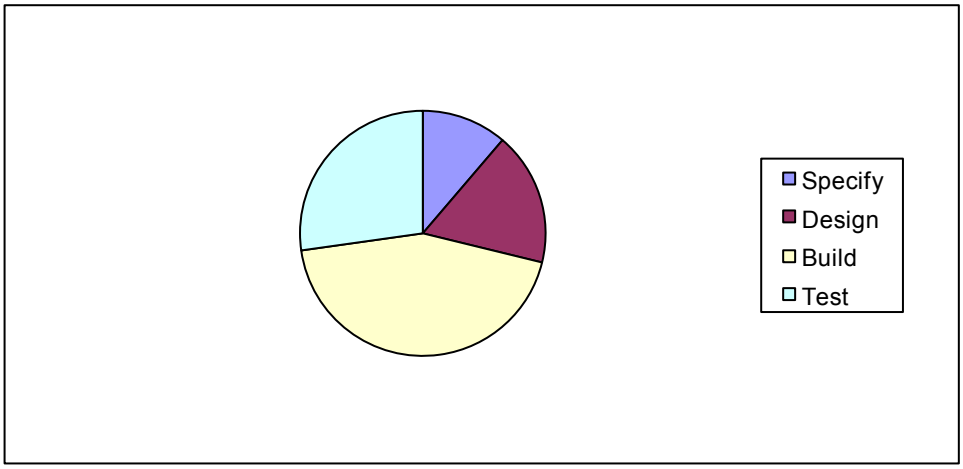


Figure 19 Business application new development projects

Ten (10) of these 35 projects also supplied the effort on Planning activities. The percentage of effort on planning for these 10 projects was very varied, with two projects reporting over 33% of effort on planning, which may indicate an error in effort recording. Excluding these two projects, the average percentage of effort on planning for the 8 projects was 5%.

The table shows the percentages depending on whether COBOL, Java or a 4GL programming language was used. Note that the percentages vary considerably for the different projects and that the numbers from which these averages are calculated are low.

Programming Language	Number of Projects	Percentage of total effort by activity			
		Specify	Design	Build	Test
COBOL	15				
Java	10				
4GL	7				

Also note that all of the COBOL projects and 7 of the 10 Java projects delivered software to run on a main-frame. The 4GL projects delivered software to run on a variety of platforms including multiple platforms.

4.3 Business application enhancement projects

Thirty-five projects supplied data on all of the effort spent on Specify, Design, Build and Test activities. This enabled the calculation of the percentages of the total effort on each of these four activities. The data were analysed in the same way as for business application new development projects, and the same warning applies about the uncertainty on the percentages due to the low statistics.

Figure 20 shows these percentages for all 35 projects and the table shows the percentages depending on whether COBOL, Java or a 4GL programming language was used.

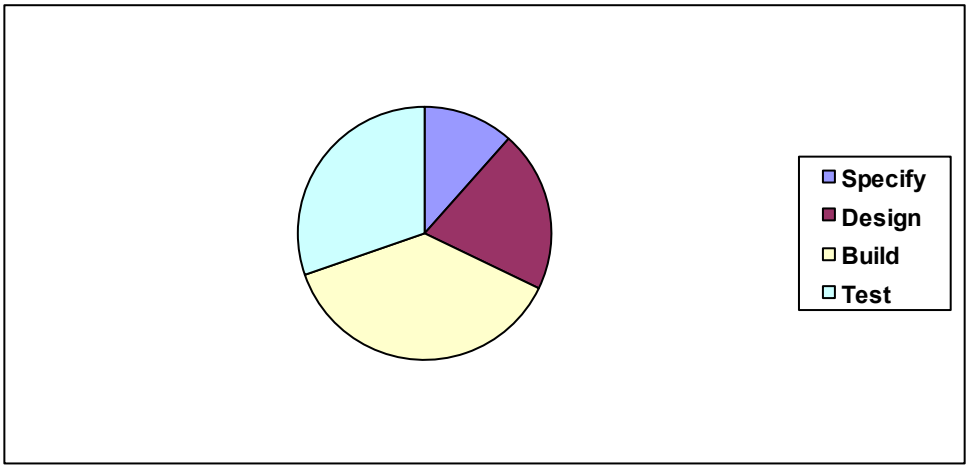


Figure 20 Business application enhancement projects: Percentage effort distribution

Programming Language	Number of Projects	Percentage of total effort by activity			
		Specify	Design	Build	Test
COBOL	22				
Java	11				
4GL	2				

These distributions, especially of the Java projects are not consistent with the expectation (e.g. from data published by ISBSG [3]) that the percentage of effort on build activities is significantly lower, and testing effort is higher, for enhancement projects compared with new developments. However, when the enhancement data for Java projects is split according to the technical platform used, as in the table below, the effort distribution for Java projects that used a main-frame is much more as expected when compared against the effort distribution for new development Java projects which also mostly used a main-frame platform.

Language - platform	Number of Projects	Percentage of total effort by activity			
		Specify	Design	Build	Test
Java – main-frame	4				
Java – PC	5				

The statistics of this last table are very low, of course. The explanation for the difference in distribution of the main-frame-based versus the PC-based projects maybe due to the technical platform or maybe due to some other factor such as the nature of the applications in the two samples. This analysis illustrates the difficulties of setting benchmarks and of drawing conclusions from small samples of project data.

4.4 The effect of the effort distribution on project productivity

There is no „correct" distribution of effort over the six activities, but evidence suggests that if the distribution of a project's effort differs significantly from the normal distribution, this is likely to be an indication that the project will experience lower productivity than if it had managed to keep close to the normal distribution.

Examples of such evidence are given in Figures 21 and 22 which show, for business application 3GL new development and enhancement projects respectively, project productivity plotted against the percentage of total project effort devoted to Specify + Design activities.

Only data from projects that reported the effort spent on both Specify and Design activities is included in these Figures. Implementation effort is excluded from the total project effort and from the productivity calculation because of its great variability. The calculation of productivity again assumes 120 work-hours per work-month.

Note certain outlier projects are not shown:
Figure 21 excludes data for two very high productivity projects with Percentage total effort on Specify & Design at 32.5% and 43.5%.

Figure 22 excludes data for three very high productivity projects, with Percentage total effort on Specify & Design at 16%, 24% and 28%.

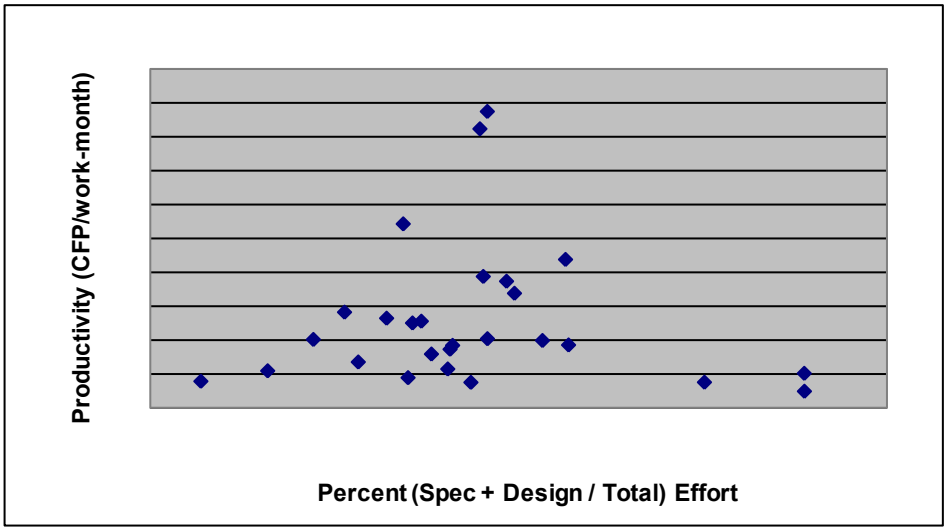


Figure 21 Business application new development 3GL projects

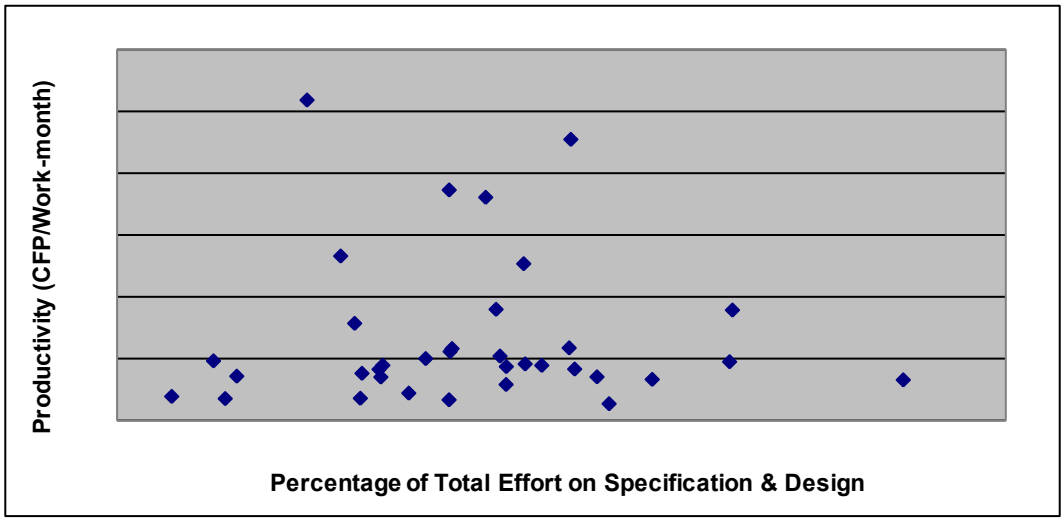


Figure 22 Business application enhancement 3GL projects

These graphs indicate that high productivity is obtained only when the percentage of total project effort devoted to Specify and Design activities lies in the range of roughly 20% - 40%. (The author has obtained similar results with other, independent data.)

This data suggests that high productivity is obtained only when the percentage of total project effort devoted to Specify and Design activities lies in the range of roughly 20% - 40%

The inferences from these findings seem to be that if the percentage of total effort on Specify and Design:

- is too low (<20%), this is insufficient effort on these two activities and productivity will be low due to re-work later in the project
- is too high (>40%), this is an indication that the project has difficulties with agreeing the specification and/or design, and has got „bogged down", causing lower productivity

(though the statistics are too low, of course, to **prove** this interpretation).

5. Some COSMIC Functional Size Characteristics

The ISBSG data collection questionnaires for COSMIC-measured projects ask for data on the composition of the total software size for each project, namely the count of functional processes and the counts of Entry, Exit, Read and Write data movements for the delivered software.

The data submitted was quite variable between the different types of software, so the statistics given here are of variable quality. Nevertheless, there are some interesting results, including those derived from the data obtained with the Finco projects (see section 3.2.4).

The tables below show two parameters for various types of software delivered in new development or re-development projects. (There is less value in showing these data for enhancement projects since the requirements for an enhancement to an existing system can be extremely variable.)

The first table shows these two parameters for two sets of business application projects, namely 34 projects from the ISBSG database and the 23 Finco projects.

Source	No. of projects	Av. no. of data movements per functional process	Average percentage distribution of data movement types			
			E's	X's	R's	W's
ISBSG	34	7.6	25%	30%	30%	16%
Finco	23	8.1	21%	39%	28%	13%

From this table we see that whilst the proportions of the parameters are similar, the percentages of Entry and Exit data movements are significantly different for the two sources. For this analysis, given that the Finco projects are all from one company and most projects delivered batch software, they will not be included in the benchmark data. The benchmark figures are therefore all based on ISBSG data, as in the next table.

Software type	No. of projects	Av. no. of DM's per FP	Average percentage distribution of data movement types			
			E's	X's	R's	W's
Business application	34	7.6				
Real-time application	4	6.7[5]				
Software component	25	2.7[6]				

These results fit a pattern that might be expected, though the low number of real-time application projects means that the figures have limited statistical validity. The average pattern is:

- Business Application software has
  a. the highest number of data movements per functional process and
  b. 55% of its functionality devoted to taking in existing data (Entries plus Reads) and 45% producing new data (Exits plus Writes)
- Real-time application software (4 projects!) devotes nearly two thirds of its functionality to taking in existing data and less than one-third to producing new data
- Software reusable components have
  a. the lowest number of data movements per functional process and
  b. equal amounts of their functionality devoted to taking in existing data and to producing new data. However, 75% of their functionality is devoted to Entries

[5] This figure is the average for only two projects that supplied the count of functional processes
[6] This figure is the average for only 8 projects that supplied the count of functional processes

and Exits with only 25% for Reads and Writes – unlike „whole" application software systems.

These average patterns are, of course, subject to wide variation for individual projects.

Some business application software may have its main task to take data in and make it persistent, thus having a relatively higher proportion of Entries and Writes, plus Reads for data validation. Some software may have as its main task to produce output and thus have a relatively high proportion of Reads and Exits. Software that must interface with many other systems will have a relatively high proportion of Entries and Exits.

Although for the Finco set of business applications the average number of data movements per functional process is only 8.1 (and the median is 7), the distribution of functional process sizes is highly skewed, with a maximum size of 70 CFP

For software components, the high proportions of Entries and Exits are to be expected given their need to interface with other components. However, there are wide fluctuations about the mean, dependent on the specialised task of the component. For example, the proportion of Write data movements in a software component varies from 0% to 47% across the 25 components analysed.

6. Comparing COSMIC benchmark data versus those of other FSM methods

Users of ISBSG benchmark data who may be starting to use the COSMIC method may wish to know how the COSMIC benchmarks compare with those with which they are familiar. In this chapter we explore the general problem of benchmark data comparisons and provide a few examples.

There are a few important general points to consider when comparing benchmark data from projects measured using the COSMIC method against benchmarks from projects using other FSM methods.

- All FSM methods measure on a relative size scale, so comparisons of absolute benchmark figures for the same set of conditions are meaningless. Only comparisons of relative benchmarks are meaningful.
  For example, comparing the ratio of the benchmark median PDR of business application new development 3GL to 4GL projects according to the COSMIC method (X/Y = 2.7 in section 3.2.2 of this report) versus the ratio according to a non-COSMIC method would be meaningful. But comparing the COSMIC figure of X for 3GL projects against an IFPUG measured figure of 12.1 [3] is meaningless.
- Non-COSMIC FSM methods for which benchmark data are available were all designed to measure business application projects, whereas the COSMIC method was designed to work for a wide range of software types. Comparisons of benchmarks will only be valid for software from the business application domain.
- Most COSMIC data used for this report has been measured within the last few years. ISBSG data based IFPUG/NESMA methods that is used as a basis for its latest benchmarks includes data dating back ten years. Comparisons of benchmarks may therefore be sensitive to evolution in project performance over the different time periods, e.g. due to using different programming languages, the need to produce web-based software, etc.
- The latest ISBSG benchmark data are based on more than 4,000 business application projects measured using the IFPUG/NESMA methods whereas the COSMIC benchmarks of this report are based on a very much smaller dataset. Users of this report should therefore take careful note of the numbers of projects used for the COSMIC benchmark data, since in some cases they are too small to be statistically very reliable.

6.1 Comparing COSMIC versus IFPUG benchmarks

There are other, specific points to consider when comparing COSMIC benchmarks against those measured with the IFPUG/NESMA FSM methods [4].

- Sizes measured according to IFPUG rules include a „Value Adjustment Factor" to account for non-functional requirements. The COSMIC method does not have such a factor. Therefore comparisons should be made only against IFPUG-measured „Unadjusted Function Points", (UFP) which is what recent ISBSG benchmarks are based on.
- On average, it has been shown [6] that there is a reasonable linear correlation of sizes of the same piece of software when measured by the COSMIC and IFPUG methods. However, there are considerable fluctuations about the average

relationship for measurements of sizes of individual pieces of software according to the two methods. These variations will be mainly due to what follows.

- The „Base Functional Components" (BFC"s) of the IFPUG method, i.e. the three „elementary process types" and the two „file types", are all measured on a scale that has artificial lower and upper size limits. For example, an elementary process can range in size from 3 to 7 UFP and a file type from 5 to 15 UFP. The COSMIC size scale for its BFC, the „functional process", has no upper size limit, whilst „file types" are not separately measured. Data in this report shows that COSMIC functional processes for business applications can range in size from 2 to 70 CFP. (For real-time software, individual functional processes have been measured of over 100 CFP.) COSMIC size measurements are therefore more sensitive to these variations in the actual amount of functionality of the components of a piece of software than are the IFPUG size measurements. At the level of aggregating the sizes of all BFC"s to a total functional size for a given piece of software, this difference in measurement approach of the two methods will sometimes be very important, sometimes less so.
- The size of an enhancement is measured quite differently on the two methods. The IFPUG method measures the size of the BFC"s of the software that are changed (i.e. added, modified or deleted). The COSMIC method measures the size of the changes to the BFC"s that are changed. The PDR for enhancement projects versus that for new development projects according to the two methods cannot therefore be properly compared.

Bearing in mind all these qualifications, a first comparison of interest is whether the IFPUG method shows the same trend for the relationship of project productivity versus size band that was obtained for COSMIC-measured projects in section 3.2.2, Figure 8.

For this comparison, data on 327 IFPUG-measured new development projects from the ISBSG Release 11 database were selected subject to the following criteria:

- Projects with data quality A or B, with size measured in Unadjusted FP by the IFPUG or NESMA methods (the latter being very similar to the IFPUG method) and using „Normalised L1 effort data"
- The „project date" was after the year 2000, from when COSMIC-measured data became available

Figure 23 shows the median productivity (in units of size per work-month, measured as above in section 3.2.2) for new development projects whose delivered software was measured by the two FSM methods, in the same size bands as in Figure 8. N.B. It is the trends of productivity with size that should be examined in Figure 23, not the absolute values. The COSMIC figures have been doubled from those in Figure 8 to help the comparison of trends on this chart.

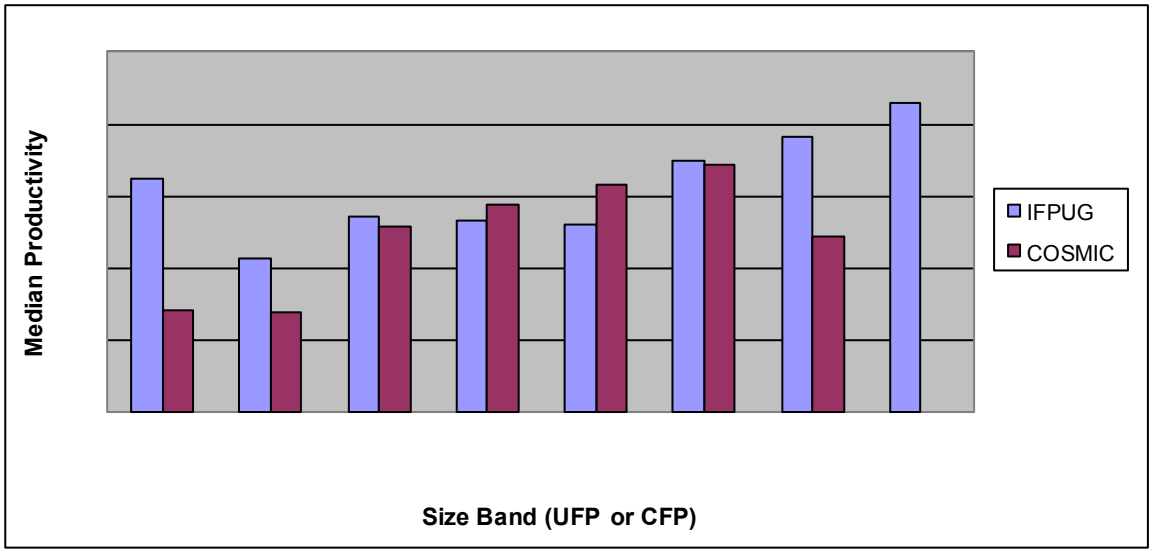


Figure 23 Comparison of trend of median productivity per size band for new development projects according to the COSMIC and IFPUG methods

This chart is quite difficult to interpret because even if the chart were based on the same projects being measured by both FSM methods, many individual projects would have appeared in different size bands according to the two methods (but this analysis used two entirely independent sets of data). Nevertheless, some observations are possible.

- Both methods show an increasing level of productivity over the size range from 50 to 1000 FP which covers 90% of all projects in the ISBSG database for both methods, which is an important economy-of-scale effect.
- The IFPUG result indicating higher productivity relative to the COSMIC result for very small software sizes is probably due to the IFPUG method's lower cut-off limits for the size of a BFC component.
- The IFPUG result indicating continuously rising productivity above 500 FP, in contrast to the COSMIC result of falling productivity above 1000 CFP may be at least in part due to the statistics (i.e. the low number of COSMIC-measured projects above 1000 CFP).

In order to help judge the significance of the data in Figure 23, the number of projects in each size band of Figure 23, on each FSM method, is given in the table below.

Method	Number of projects in each size band							
	Band 1	Band 2	etc					
COSMIC	18	30	17	23	14	17	7 [7]	0
IFPUG	9	36	47	58	67	51	41	18

(When the median productivity of enhancement projects per size band for IFPUG-measured projects (from ISBSG Release 11 data) is compared against the COSMIC data as in Figure 14, the trend of increasing productivity with size is practically identical – this in spite of the two methods measuring different types of sizes for an enhancement project.)

Understanding the relationship between software sizes as measured on the two methods is therefore both difficult and important. Given that both methods show some variation of productivity with size of software delivered, it is important to take into account differences in

[7] The maximum size of a COSMIC-measured project in this band is 1670 CFP

size mix when comparing any two benchmark figures on any one FSM method, or across methods. Standard ISBSG benchmark analyses sub-divide project data on only one parameter at a time (apart from splitting data across technical platform) and do not consider the effect of size mix on PDR.

The next table gives two comparisons of the ratio of median benchmark PDR figures for new development projects, measured on the two FSM methods, where the IFPUG data is taken from [3] for „All projects", i.e. for projects irrespective of hardware platform. IFPUG sizes are measured in Unadjusted Function Points. N.B., a ratio > 1.0 indicates lower productivity. The COSMIC benchmark ratios have been corrected for size mix effects as described in section 3.2.5

Ratio of PDR Benchmarks	COSMIC		IFPUG	
	New Devts.	Enhance-ments	New Devts.	Enhance-ments
3GL projects / 4GL projects	2.7	1.0	1.4	1.4
COBOL projects / Java projects	1.2	1.0	2.1	2.2

It is not obvious at all why these ratios apparently differ according to the two methods.. (The numbers of COSMIC projects used in these two comparisons are sufficient that the explanation should not be due to limited statistics.) To explain the differences would require a much more detailed analysis of the data than time has permitted. The explanation might lie in the mixes of projects used for the data on the two methods being different due to:

- COSMIC data are for business application projects; IFPUG data is for „all projects" which include in the order of 10% of projects concerned with real-time software
- the technology platforms used
- the mix of software sizes delivered
- the age of the projects (COSMIC data are mostly from the last 3 years, IFPUG data are for projects that used IFPUG 4.0 or higher method, going back 15 years

However, an overall conclusion from the findings of Figure 23 and the table above is that the COSMIC-measured project data seems to give more differentiated benchmarks for different environments than do the IFPUG-measured projects, even after allowing for the less differentiated analyses carried out by ISBSG on the IFPUG data.

6.2 Comparing COSMIC versus MkII FPA benchmarks

The MkII Function Point Analysis (FPA) method is very similar in design to the COSMIC method. Both methods define functional processes as their BFC, though the MkII FPA method uses other parameters than data movements to size the input, process and output components of each functional process. Like the COSMIC method, the MkII FPA method has no upper limit to the size of a functional process. The MkII FPA approach to sizing enhancements is the same as for the COSMIC method, i.e. both methods measure the size of the changes to software required in an enhancement project. Given this similarity of FSM method design, one would expect greater similarity of benchmark findings between the COSMIC and MkII FPA methods than between the COSMIC and IFPUG methods.

The only MkII FPA benchmark data available to the author dates from 1996 [7], based on measurements of over 600 business application projects, so the data is now only of historic interest. However, it is worth noting the following from a brief comparison with the COSMIC benchmark data in this report.

- The variation of productivity with the size of software delivered by new development projects is roughly the same when measured by both methods
- The ratio of the PDR for 3GL new development projects to that for 4GL projects on the MkII FPA method is about 3 over the whole size range (COSMIC = 2.7)
- The ratio of the PDR for 3GL enhancement projects to that for 3GL new development projects of the same size on the MkII FPA method is about 1.4 (COSMIC = 1.1)
- The ratio of the PDR for batch projects to on-line projects is about 1.7 for both methods

These comparisons are subject to several uncertainties but, on a range of parameters, the MkII FPA and COSMIC methods give similar benchmark ratios, which is to be expected given the similar design of the two measurement methods.

7. Summary Findings and Conclusions

The ISBSG repository of data for software new development and enhancement projects now contains close to 350 projects that have been sized using the COSMIC method and the number submitted to the ISBSG continues to grow steadily. The number of projects is now adequate to publish meaningful benchmark data on several parameters, though the reader is cautioned in several places in this report that some of the benchmarks are based on very small numbers of projects.

Some of the key findings from this study are as follows.

a) Figure 24 shows the median PDR figures for five main project categories.

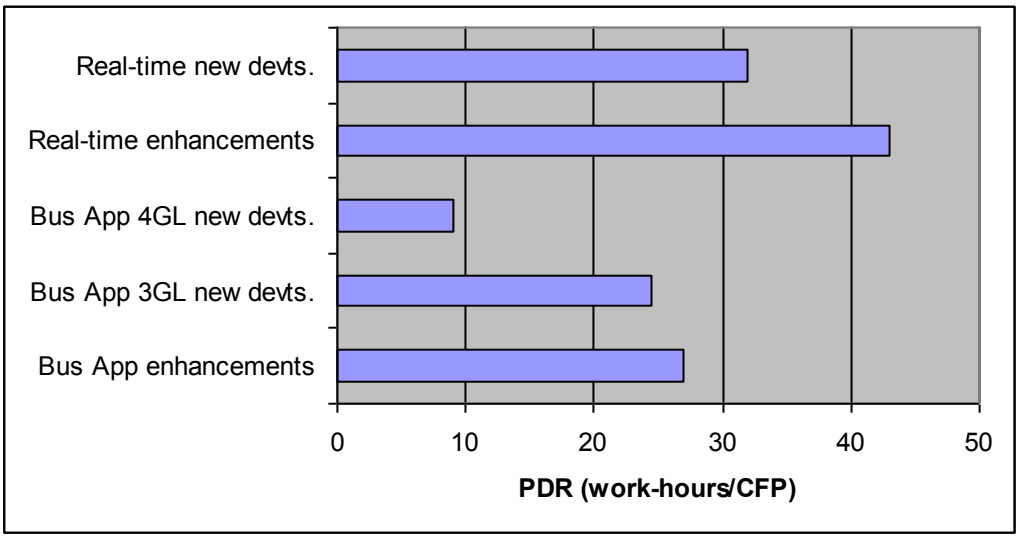


Figure 24 Key PDR benchmarks

These results are unsurprising:

- Real-time projects require more effort per CFP than do business application projects
- Enhancement projects need more effort per CFP than new development projects
- Projects developed using a 4GL require much less effort than those using a 3GL

b) The PDR for developing software reusable components is typically at least an order of magnitude lower (i.e. higher productivity) than for developing whole application systems. This means it is imperative not to mix measurements of the output of projects that deliver whole software systems and those that deliver small components. (The size of a „whole" software system cannot be obtained by adding up the size of its components – see the rules of the COSMIC method). The ISBSG data collection questionnaires for COSMIC-measured projects will be upgraded shortly to ensure that this distinction is more clearly made at the time of data submission.

c) Median productivity for business application new development projects increases with size of the delivered software, more than doubling for projects delivering in the X (highest) CFP range compared with projects delivering less than Y (lowest) CFP. Above 1000 CFP, the data suggests that productivity falls, though there is considerable statistical uncertainty on this point.

d) Median productivity for business application enhancement projects rises continuously with increasing size of the delivered software, more than doubling for projects delivering in the X (highest) CFP range compared with projects delivering less than Y (lowest) CFP.

e) The speed of projects (measured as CFP per elapsed month) increases sharply with increasing size of software delivered for all categories of projects. Power curves fitted to the data, of the form Duration = C x (Effort)^n, have an exponent „n" for new development projects of A and for enhancement projects of B. This finding is consistent with the old rule of thumb that the elapsed time for a project increases roughly with the square root of the effort of the software to be delivered.

f) The increasing project productivity and speed with increasing size of software delivered show a clear economy-of-scale though, of course, this benefit must be offset by the increasing risk associated with larger projects. This increasing risk is very well illustrated by the data of Figures 9 and 15 which both show an increasing spread of productivity with increasing size.

g) Comparing Figures 9 and 15 also shows that the scatter of productivity about the median is significantly broader for enhancement projects than for new development projects over the whole size range. This is at least partly explained by the varying nature of enhancements that must be carried out.

h) Given that for new developments, the median productivity of projects using a 4GL language is 2.7 times greater than for projects using a 3GL language and that 4GL projects are delivered faster, the gains on both productivity and speed from using a 4GL versus a 3GL for new developments are very significant. However, projects to enhance software that was developed using a 4GL are no more productive than projects to enhance 3GL software.

i) New development projects that must deliver software to run in batch mode require about X (where X> 1) times as much effort per CFP as do projects that must deliver software to run in on-line mode. The COSMIC-measured statistics are weak on this result, but it is noteworthy that this same result was observed many years ago on projects where the software size was measured using MkII FPA.

j) The effort distribution over project activities varies with the type of project. Comparing the distributions of effort of Figures 19 and 20 (and taking into account the more detailed analyses), we see that enhancement projects spend proportionately more effort on testing and less on build activities than new development projects. The higher proportion of effort on testing, usually including regression testing, for a given amount of changed or added functionality, may largely explain why enhancement projects are less productive than new development projects.

k) The effort distribution data suggest that spending relatively too little (< X%) or too high (> Y%) a proportion of total project effort on specify and design activities will result in lower productivity. It is therefore important to take into account these norm distributions of effort when planning a project.

l) The report includes average ratios of numbers of data movements per functional process, and distributions of the average percentages of Entry, Exit, Read and Write data movements for different types of software. These data can be useful for checking the credibility of measurements of functional size using the COSMIC method for these various software types.

m) The report discusses the general problems of comparing benchmark data obtained from projects where the software size is measured using the COSMIC method versus other functional size measurement methods. Some specific problems are identified on comparing COSMIC against IFPUG benchmarks as published by the ISBSG. Some examples of such comparisons show similar trends (e.g. increasing productivity with size of software delivered) but others suggest that the COSMIC method gives more differentiated benchmark figures for differing sets of project characteristics. However, some of this apparent difference may be due to the way that the ISBSG analyses its project data, which mostly cut the data on only one parameter at a time. (Example ISBSG produces benchmark PDR figures for new development and for enhancement projects and then, separately, for projects using a 3GL or 4GL programming language. In this report we have attempted to derive PDR figures for all four combinations of these two parameters where there are sufficient data).

Overall, the COSMIC benchmark data reported here seems to be nicely self-consistent in spite of the limited statistics in some areas. The data should be valuable for organizations wishing to make external performance comparisons and for new project estimating.

It would be great to have more data. Users of the COSMIC method are strongly urged to submit their project data to the ISBSG database so that the accuracy and scope of the benchmark data can be improved.

The COSMIC Functional Size Measurement Method

Version 3.0.1

Guideline for assuring the accuracy of measurements

VERSION 1.0

February 2011

Acknowledgements

Version 1.0 authors and reviewers 2011 (alphabetical order)		
Alain Abran, École de Technologie Supérieure, Université du Québec, Canada	Manfred Bundschuh, University of Applied Sciences Cologne, Germany	Jean-Marc Desharnais*, École de Technologie Supérieure, Université du Québec, Canada
Peter Fagg, Pentad Ltd., UK	Arlan Lesterhuis*, The Netherlands	Marie O'Neill, Software Management Methods, Ireland
Grant Rule, Software Measurement Services, United Kingdom*	Luca Santillo, Agile Metrics, Italy	Charles Symons*, United Kingdom
Frank Vogelezang, Ordina, The Netherlands		

* Authors of this Guideline

Foreword

Purpose of the Guideline

This Guideline provides an overview of the factors that influence the quality of a functional size measurement, particularly a COSMIC size measurement, and advises how to organize the measurement process so as to obtain accurate size measurements and/or to improve measurement accuracy.

Quality assurance actions aim to improve quality and to help achieve desired quality levels. In our case a measurer's quality aim will be to achieve a desired measurement accuracy level.

The desired accuracy of a measurement (see below for the definition of 'accuracy') will probably vary with the project circumstances. Early in the life of a new software project, only an approximate estimate of size may be possible, but it may still be of acceptable accuracy. At the other extreme, e.g. in the context of outsourced software contracts where a measurement may at some time become a matter of legal dispute, highly accurate measurements are likely to be essential.

In order to assure the desired accuracy of measurements, it is important to have both well-trained and experienced measurers and an adequate and repeatable measurement process. Audits are also necessary to check and to assure on-going accuracy.

In managing the measurement process, two main quality control efforts are essential. They are, in order of importance:

- Error-prevention,
- Defect-detection.

Error-prevention efforts concern three main factors[1], namely assuring the:

- Quality of the measurer(s)
- Quality of the software artifacts
- Quality of the measurement process

Defect-detection that only *identifies* defects may not help much to prevent their re-occurrence. Rather, audits should identify defects *and* determine the causes of the defects that have been made when measuring. Systematically determining the root conditions and analyzing the causes of defects will encourage those responsible to improve the measurement process and in future avoid time wasted in audits and defect corrections.

Intended Readership of the Guideline

This Guideline is intended to be used by measurers and by the management of measurers, who have the task of organizing functional size measurement.

Readers of this Guideline are assumed to be familiar with the COSMIC Method.

Introduction to the contents of the Guideline

Chapter 1 discusses the error-prevention features of the measurement process. Chapter 2 discusses defect-detection, auditing measurements and of the measurement process. Chapter 3 presents a simple audit or 'self-audit' checklist that may be helpful to indicate the quality of a measurement. This

[1] The factors correspond with the sections in chapter 1. The order in which the factors appear does not imply an order of significance.

checklist can be used, for example, by anyone submitting COSMIC-measured project data to the ISBSG repository to give a quality-score for the measurement.

Conventions and terminology used in the Guideline

- Quality control and assurance efforts require a number of actions to be performed. Actions and points of interest are indicated by a '>' sign.
- There are many kinds of documentation and other artifacts used for measuring or estimating functional size, such as functional user requirements (FUR), designs, definition studies or even the software itself (screens, reports, etc). In this Guideline these are all referred to as 'software artifacts'. Where appropriate, the reader of this Guideline must judge which one (documentation, software or both) applies.
- 'Measurement accuracy' is defined as 'the closeness of agreement between a measured quantity value and a true quantity value of a measurand'[2].
- A 'defect' is defined as a product anomaly[3]. A defect is introduced into a work-product when an error or mistake is made in a process.
- 'Inspection' is a close examination of something. 'Auditing' is the verification of something with respect to a documented procedure. Where there is no documented procedure, read 'inspection' where this Guideline mentions 'audit'.

[2] The ISO-IEC Guide 99: 'International vocabulary of metrology – Basic and general concepts and associated terms (VIM)', Latest edition 2007.

[3] From IEEE 982.1-1988 IEEE Standard Dictionary of Measures to Produce Reliable Software.2.00.

Table of Contents

1

ERROR-PREVENTION

1.1 Quality of the measurers

The quality of the measurers is determined by their knowledge of the COSMIC Method and by their experience in measurement. This quality factor pertains to 'who' is performing the measurement.

> Measurements may be carried out by a 'measurement team', i.e. a team of dedicated measurers, or by (some of the) developers. However measurement is organized, it is crucial that the measurers can obtain and maintain sufficient experience. For this same reason it is advised against having persons measure infrequently without a strong verification process to support them.

> Measurers should be trained in and if possible should be certified for the COSMIC Method.

> Periodical COSMIC measurement workshops should be performed for refreshing measurement knowledge and exchanging experience. Periodically requiring all measurers to measure the same piece of software in a 'blind' or 'double-blind test'[4] provides a good way of checking competence, mutual consistency of the measurers and of identifying specific measurement problems.

> Audits of measurements should be performed periodically in order to identify conformance to the mandated process, defects arising and variations in approach (see chapter 2).

> Measurers should regularly check on www.cosmicon.com for the availability and relevance of 'Method Update Bulletins' (MUB's) and any discussion of measurement issues.

1.2 Quality of the software artifacts

The quality of the artifacts to be measured is one of the important factors which determine the accuracy of the size measurement. Given the range of artifacts that may have to be used for measurement, from diagrams, to text, to installed software, it is impossible for the COSMIC method to define procedures for the extraction of the Functional User Requirements needed for measurement. This quality factor pertains to 'what' is the subject of the measurement, i.e. the information source(s) of the measurement.

The authors of software artifacts may not know which aspects must be made clear for COSMIC sizing. Therefore the following recommendations are made to help obtain good quality artifacts.

> Authors of software artifacts and those who will use them should be knowledgeable about the generalities of Functional Size Measurement and its uses and, at least in outline, knowledgeable about the COSMIC Method. The organisation will benefit from such an investment in know-how because estimates and measurements will be made earlier, more consistently, be more repeatable, with fewer defects, necessitating less rework. (Experience shows that artifacts that cannot be measured almost certainly have defects such as omissions, ambiguities, etc.

[4] In a 'blind' test the measurers do not know that a test measurement is being held, the distributor of the measurement does. In a double-'blind' test, neither the measurers nor the distributor of the measurement know that a test measurement is being held.

Measuring using the COSMIC method therefore provides a valuable quality control of the software artifacts. See [12]).

> Local instructions should be issued to advise authors how to express software artifacts in a form that is suitable for COSMIC size measurement and/or to adapt software artifacts for measurement purposes. In particular, it should be easy to identify in the software artifacts the central concepts of the COSMIC Method. A few simple examples of what the measurer means with these concepts should encourage authors to capture the required aspects. The measurers should offer assistance to the authors in capturing these aspects, if desired.

> Controlled tests have often shown that the most common errors made by measurers are to miss (i.e. fail to recognise) functionality that should be measured, so that measured sizes are often less than the actual size. Inexperienced measurers miss more functions than experienced measurers. Measurers therefore should check particularly that the software artifacts they are given represent the best knowledge available at the time and are as complete as possible.

> Take particular care early in the life of a software product when the requirements are not yet known in all the detail needed for an accurate measurement and when they may not be stable. When software artifacts are defined coarsely, at a 'high level', approximation variants of the standard COSMIC method may be used (see the 'Advanced and Related Topics' document [3], Chapter 1, available from www.cosmicon.com.). **At all times, but especially early in the life of a software product, it is valuable to estimate and to report the uncertainty in a measured size. This is achieved by expressing the estimate as a range.**

EXAMPLE 1. A range can be determined for instance as follows. Given a software artifact, identify the parts a) that can be measured accurately and b) that cannot be measured accurately. Measure the size of the former parts, resulting in say S CFP. Of the latter parts, estimate the lower and the upper limit of the size (say L respectively U CFP). The true size is then expected to be between S+L and S+U CFP.

EXAMPLE 2. A software contract specifies that at a defined stage early in the life of a project, the functional size must be measured to an accuracy of 'within + or – 20% of the true value'. A measurer is given the task of measuring a certain statement of FUR and concludes that their quality is such that it is not possible to achieve an accuracy of better than + or – 30% of the true size. Hence, the measurer refers the matter to the customer and/or project manager so that either the contractual terms can be adjusted, or the quality of the FUR can be improved.

EXAMPLE 3. An example statement of FUR is produced to be as clear and unambiguous as possible, for use in a certification examination for candidate measurers. The examination board commissions a group of experts to measure the FUR, and to take the mean of their respective sizes to be the 'expected answer' (the nearest the experts can agree on a 'true size'), which happens to be 260 CFP. Taking into account the variation in the measurements made by the experts and the time pressures of an examination, the examination board agrees that any candidate providing a size within the range 250 to 270 CFP will be considered to have produced a sufficiently accurate answer and will pass this test.

> Checks of the quality of a software artifact should be made. See Appendix A for checklists.

1.3 Quality of the measurement process

A document describing an organization's measurement process should provide both an overview to the organization and the detail needed by measurers. This quality factor pertains to 'how' the measurements should be performed.

> Design and describe the desired measurement process. Aspects to be considered are

- The policies for measurement, i.e. purpose of measuring, stakeholders and their benefits.
- The organization of the measurement process: its activities, who initiates, who is involved, expected effort of the involvement, what data is registered, which artifacts produced for whom, and their expected reaction
- At what stage(s) of the development process measurements are required, on which work-products, how they will be used, the minimum standards expected for their accuracy and whether and why measurements are mandatory

- Local variations on the standard COSMIC measurement process, if applicable.
- The steps of each measurement activity and the tools and facilities to be used, with an explanation
- The allocation of responsibilities for measurement data capture and its quality, the organization, maintenance and use of the repository
- The intended uses of the measurement data, e.g. for improving organizational performance, estimating, etc. Requirements for the presentation of the results of measurements, respecting the interests of the various stakeholders in the measurement activities

> As there are many possible ways in which requirements are expressed, there are no general rules for mapping software artifacts to the COSMIC concepts. Therefore, the way of mapping the local types of software artifacts should be documented.

> For each measurement, a Measurement Report should be drawn up. This can be very simple, but should capture the Measurement Strategy parameters as well as the measurement results. Also, measurers should record defects and ambiguities found in the software artifacts as well as any assumptions made from the artifacts. Such reports will serve as the basis for audits and for future measurement process improvement activities. For details, see chapter 5, 'Measurement Reporting', of the Measurement Manual [1].

> Data concerning the measurement process are a source for improvement. Therefore results of the COSMIC measurements should be adequately registered. For details, see section 'Archiving COSMIC measurement results' in chapter 5, 'Measurement Reporting', of the Measurement Manual.

> Measurement data may be stored either centrally (one central measurement data repository) or de-centrally (a number of local collections of measurement data). It must be considered carefully how to organize the storage of the measurement data, taking into account future use of the data. If data are stored de-centrally, procedures must ensure compatibility of the data across the de-central repositories.

> For each measurement, the Measurement Report and the related software artifacts should be stored in such a way as to preserve their relationship. These documents serve as 'witnesses' of the measurement and enable audits afterwards. They establish the audit trail that is beneficial during subsequent enhancement and support and maintenance activities.

> The measurement process should aim to ensure traceable version control of measurement (input) documents. For instance, to ensure that the latest available documents are used for the final Measurement Report, rather than preliminary documents.

> Several tools exist to support the measurement process, ranging from simple spreadsheets to more sophisticated tools for registering measurement data and relating them to the software artifacts used (see www.cosmicon.com for examples). More sophisticated tools exist that combine capturing measurement-data and project planning or for capturing functional user requirements which can be measured automatically. However, the quality of these tools should be verified, especially the quality of non-transparent tools. No tool should be used that is not fully understood by its users. Otherwise, defects of the tool itself and errors made during use of the tool will not be noticed.

> When a (supposed) shortcoming of the COSMIC Method has been encountered, a Comment or Change Request should be submitted as per the process described in Appendix B. The editors of the COSMIC Method will review the Comment or Change Request and issue a solution to a shortcoming, if appropriate, e.g. a Method Update Bulletin, as quickly as possible.

2

AUDITING: DEFECT-DETECTION

The purposes of auditing can be

- to determine the accuracy of a particular measurement, by revealing defects
- to determine how well the measurer has conformed to the measurement process, and to identify opportunities to improve the measurer's skill, experience and inter-measurer consistency
- to verify and validate the measurement process, its effectiveness, cost and timeliness, so that it can be improved

It is common for defects in a measurement to have been caused by defects in the software artifacts, and/or by errors by the measurer in mapping from the artifacts to COSMIC concepts. For example, vagueness or lack of clarity of the software artifacts may lead the measurer to make wrong assumptions and interpretations (implicit or explicit), leading to defects in the measurement. Whatever the cause may be, a very important point is that with a well-organized process for recording and processing any defects found, the *causes* of defects can be analysed, enabling the organization to take action to prevent future defects of similar type, in priority order of their importance and/or frequency. Besides, it enables both auditors and measurers to judge themselves and identify their strong and weak points.

Research has shown that performing functional size measurement itself can be helpful in detecting defects in software artifacts, by revealing omissions or inconsistencies in functional processes that are difficult to detect with regular audit procedures [14].

See also the Checklists in Appendix A which are designed to assist defect detection.

2.1 Auditing of measurements

A complete audit of a particular measurement may have three components.

- An audit of *methodological correctness*: it is usually carried out by a fellow-measurer.
- An audit to verify the *accuracy* of the measurement of the given software artifacts; it is usually carried out by the author of the artifact or a system expert, working with the measurer to explain COSMIC details.
- An audit of the *measurement documentation*; it is usually carried out by a fellow-measurer and may take no more than a few minutes.

> Methodological correctness. The purpose of the first component is to verify, preferably on the basis of a random sample of the software artifacts[5], that the measurement complies with the requirements of the COSMIC Method. After correction, the measurement should be 'methodologically correct'.
>
> After the first component, the fellow measurer and the measurer should discuss the (supposed) defects. When both agree on some defect[6], its cause can be analysed. This information is

[5] Do not use a sample of the *measured* functionality, as in that case functionality that has been forgotten to be measured will not be detected.

[6] Check the (supposed) defects found by the auditor, an auditor can also make mistakes!

captured. When auditor and measurer differ in opinion about the cause of the defect, both causes should be recorded.

The defects found and their causes should be registered and may be categorized as shown in the table below. The findings and a Pareto analysis of the results (i.e. the causes in decreasing number of errors per category) should be presented periodically to the measurers and other stakeholders and should be the basis of recommendations and/or actions.

Defect Cause (to be entered by the auditor)	**Explanation**	**Possible action to prevent re-occurrence**
Measurer	The measurer knows how the situation should have been measured but made a mistake, or the measurer doesn't know how to measure a situation although the COSMIC Method deals with it	Periodical workshops (see section 1.1) and/or train the measurer
Software artifact	The software artifact describes a situation admitting of more than one interpretation, perhaps leading the auditor and the measurer to different solutions	Require the author to correct the defect and re-measure. Draw up requirements for software artifacts to be measurable
Measurement process	Any aspect of the measurement process, (e.g. a local rule for mapping from FUR to COSMIC concepts admits of more than one interpretation)	Depends on the aspect. If the cause is the local set of mapping rules, make the necessary improvements

> Measurement accuracy. The purpose of the second component is to verify that the measurement faithfully represents the functionality. For this, the measurement should be discussed with the author of the software artifacts that have been measured (or with another functional expert representative of the commissioner of the measurement). Their first task is to ensure that the measured software artifacts are still the correct ones[7]. Subsequently, parts of the artifacts that are not clear to the measurer must be clarified by the author and the measurer's assumptions and interpretations must be verified. Finally, the author must verify that no (change of) functionality has been neglected, nor that functionality has been included in the measurement that is not within the scope. After correction, and when the corrected measurement has been checked by the author, it should be 'functionally correct'. Relevant data can now be registered. Defects found should be recorded as per the table above, actions taken and analysed as per the first component.

> Measurement documentation adequacy. The purpose of the third component is to verify that the measurement is correctly finished. For instance, are data correctly registered, are all relevant documents present, have superfluous documents been removed? It should be carried out by a fellow-measurer at the end of the measurement activities. This component may seem trivial but has been found to be very effective in practice. For, once registered a defect won't be noticed as such anymore, resulting in confusing reports and calculations. Moreover, if a defect is noticed long after registration it is often difficult to reconstruct the original situation and repair it, thus undermining the reliability of the measurement and the output based on the data.

2.2 Auditing of the measurers

Besides the audits described in the previous section, the measurers may expect occasional audits to assess their quality, for a number of purposes. For these audits a well-organized registration and

[7] In practice, sometimes software artifacts have been added, removed or changed without the measurers having been informed.

document storage is indispensable, because it enables the measurers to supply a sample of documents, as specified by the auditors, to be audited and/or re-measured.

> Periodically (once or twice a year) an audit may be held on behalf of the management of the measurers. The object of this audit is to assess the quality of the measurers. Ideally it should be held by an independent and certified third party. The audit consists of re-measuring a sample of the measurements for a selected period. Such audits are especially held in situations of outsourced software development, when payments depend on functional sizes, but may also be held for 'training needs analysis'.

2.3 Root cause analysis of defects

If certain types of defects continue to arise repeatedly, in spite of taking the recommended remedial action, then an analysis of the 'root cause' will become necessary to assure the quality of the measurements. Examples might be:

- The artifacts made available for measurement continue to be of poor quality. Action may be needed to improve the organization's documentation standards (to meet measurement needs) or from higher management to enforce their observance
- The training given to measurers and/or to those who produce documentation needed for measurement may be inadequate, requiring either improvement of the training material (e.g. by adding better case studies) or action by the trainers to improve their presentation skills.

3

DETERMINING THE QUALITY RATING OF A MEASUREMENT

All COSMIC functional size measurements should be accompanied by a quality rating[8]. The rating method presented in this chapter is pragmatic, is quick and easy to use and has been applied successfully in practice. The rating level, which should ideally be at level 'Good' or 'OK', gives an indication of the probable accuracy of the measurement (see below).

To determine the quality rating of a measurement, use the following steps (by means of the self-audit checklist below):

- Assign a rating of 'Good', 'OK', or 'Poor' to each of the three quality aspects
- The quality rating of the measurement is the **lowest** of the three scores

EXAMPLE. If the ratings of the three aspects are 'Good', 'Good' and 'Poor', then the quality rating of the measurement is 'Poor'.

When a functional size measurement is based on either a) an approximate method or b) on a known incomplete software artifact the functional size measurement is assigned a rating 'Approximate'. Note that measurements with this rating may be suitable for purposes where rough estimates suffice.

3.1 The self-audit checklist

Quality Aspect and Ratings	**System ID:**
1. Quality of the measurer(s) Good = Entry-level certified and >2 years experience in COSMIC measurement or 10,000 CFP measured; measures regularly OK = Entry-level certified AND measures at least once per month* Poor = Beginner OR measures only occasionally* * But if the measurement is audited by an independent, experienced measurer assign 'Good'	
2. Quality of the software artifacts[9] Good = Excellent documentation; access to the actual system AND to a system expert OK = Reasonable documentation AND access to the actual system OR to a system expert Poor = Limited or poor documentation; no access to the actual system or to a system expert	
3. Quality of the measurement process Good = Measurement records show traceability of the functional processes and data movement types (E, X, R, W) to the software artifacts, on at least a standard spreadsheet; other	

[8] The approach sketched in this chapter is aligned with the rating method used for measurements submitted to ISBSG (International Software Benchmarking Standards Group, a global and independent repository and source of data and analysis for the IT industry), and if used might therefore facilitate the collection, rating and submission of COSMIC-measured project data to ISBSG. ISBSG uses the levels A, B and C with A = 'Good', B = 'OK' and C = 'Poor'.

[9] A detailed method of assessing the rating for this aspect is given in section 3.2

		measurement parameters are well documented. There was sufficient time for proper measurement and recording
OK	=	Standard measurement process. The measurement is recorded at least to the functional process level with references to the software artifacts
Poor	=	Informal (undocumented), or individual process, or measurement under severe time pressure; measurement results are not clearly traceable to the software artifacts.

For information

There are no data on how the accuracy of a measurement varies with the quality rating; an initial guess is that the variation is roughly as follows.

Good	= within 5% of the true size
OK	= within 10% of the true size
Poor	= unknown accuracy
Approximate	= depends on the approximation method used

Note. One of the commonest measurement errors is to miss some functionality so that the measured size is lower than the true size. Claiming an accuracy of 'within 10%' may not, therefore, be the same as claiming that the result is 'within plus or minus 10%' of the true size.

3.2 Rating the quality of software artifacts

This section describes a simple process to obtain the rating (Good, OK or Poor) of aspect 2 for a set of software artifacts (the 'software artifacts rating'), in a way that is less subjective than just making a single judgment based on the criteria given in the table above. This process might be applied for instance to support a benchmarking process rating, e.g. if the measure being examined is to be submitted to ISBSG repositories. The software artifacts rating is obtained by requiring the rating to be traced directly to the software artifacts used for the measurement.

The software artifacts rating is determined in two steps. In the first step the functional processes of the software artifacts are rated, leading to a number of functional process ratings (sections 3.2.1 and 3.2.2). In the second step the functional process ratings are used to derive the software artifacts rating (section 3.2.3).

Note. Capturing the functional process ratings at the time of the measurement requires very little extra effort.

3.2.1 Determining the functional process ratings

Rate each functional process identified in the set of software artifacts on the scale (a) to (e) as follows:

(a) The functional process is completely documented

(b) The functional process is documented but the description of the data moved is unclear

(c) The functional process is identified only

(d) The number of the functional processes is given but they are not specified

(e) The functional process is not mentioned in the artifacts but is implicit

Explanation of the functional process ratings:

(a) Each functional process is fully documented, together with its data movements by type (Entry, Exit, Read and Write) including the objects of interest and their attributes.

EXAMPLE of a FUR. 'The system will produce a monthly file. The interface format details are provided in the Interface ID v2.8 in an Excel format document and the entity relationship diagram that shows each attribute is in the ER.gif document version 2.1.'

(b) Each functional process is documented. The input, output, stores and retrievals of each functional process are also described but not clearly enough to identify the number of data movements

EXAMPLE of a FUR. A yearly report will be provided to the manager with the following information: administration category, name of the administration, origin, amount of sales by month, type of product and sales by product, etc. (In this case the reference to the data groups is not clear.)

(c) Functional processes can be identified but not their data movements

EXAMPLE of a FUR. The sales representative will provide a monthly report to the manager.

(d) The number of the functional processes is given but they are not specified.

EXAMPLE of a FUR. Five reports will be provided to the manager.

(e) The functional process is not mentioned in the artifacts but is implicit

EXAMPLE. A screen that allows modification of data about an object of interest within a functional process usually requires the display of the information before modifying the data. Sometimes this display requirement is not documented.

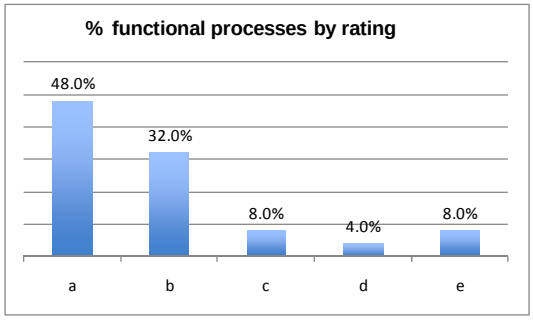
Note 1. A functional process rating may be increased if access to the actual system and/or to a system expert can provide reliable information that is not available in the artifacts.

Note 2. To promote good documentation, it may be helpful to record two functional process ratings: one using the documentation only, and the other taking into account information from an expert and/or from examining the actual system. For ISBSG data submission, only the latter rating should be considered.

3.2.2 Presentation of the functional process ratings

The distribution of the quality ratings for the documentation could be provided based on the percentage of functional processes in each rating. The table shows an example of the rating of a set of software artifacts describing 50 functional processes (from a real set of FUR) and the results are also shown graphically.

Rating	Number of functional processes	Percentage
a	24	48,0%
b	16	32,0%
c	4	8,0%
d	2	4,0%
e	4	8,0%
Total	50	100,0%



3.2.3 Interpretation of the functional process ratings and deriving the software artifacts rating

From these results, the software artifacts rating should be given as ‘Good’, ‘OK’ or ‘Poor’. The following is a suggested way of deriving this quality rating

- A result with more than 70% of the functional processes rated as (a) should be considered for a software artifacts rating of ‘Good’.
- A result of 80% or more of the functional processes rated as (a) or (b), of which at least 50% (a) should also be considered for a software artifacts rating of ‘Good’.
- A result with 60% or more of the functional processes rated as (a) or (b) and at least 30% (a) should be considered for a software artifacts rating of ‘OK’.
- A result with 70% or more of the functional processes rated as (b) and less than 10% rated as (a) should be considered for a software artifacts rating of ‘OK’.
- A result with less than 50% of the functional processes rated as (a) or (b), and less than 10% rated as (a) should be considered for a software artifacts rating of ‘Poor’.

With 5 functional process ratings there are 31 possibilities[10], so the measurer will need to use some judgment when applying these suggestions.

EXAMPLE. The example results given in the table and graph above, with 48% (a) and 32% (b), could lead to a software artifacts rating of

- 'Good', applying the suggestion of the second bullet or
- 'OK', applying the suggestion of the third bullet

Because 48% is not too far from 50% and the total is 80% the software artifacts rating could be 'Good'.

The rating method could be further refined by taking into account the size of the functional processes as well as their percentages. So the rating of larger functional processes would carry more weight than smaller functional processes.

[10] The possibilities correspond with the 31 non-empty subsets of the set {a, b, c, d, e}.

APPENDIX A – CHECKLISTS

The three checklists in this appendix can be used together or separately. The first checklist concerns the form and content of the software artifacts. The second checklist can be used to enable a systematic check of measurements. The last checklist focuses on areas where errors are commonly made and may be used when time is limited.

A.1 Checks on the quality of software artifacts

> Do all software artifacts of one organization presented to the measurers (in one measurement or in measurements over time) conform to local standards and have the same format? The same content (i.e. the same way of specifying the events, functional processes, the objects of interest)? The same way of diagramming data models or other concepts? The same level of granularity? Do they describe software at the same level of decomposition? (A lack of consistency of documentation across the organization may be an indicator of quality problems.)

> Does each software artifact describe the purpose of the software and the (business) process that it must support?

> Does each software artifact contain all the information that the measurer needs for the COSMIC measurement (can events, functional processes etc. easily be identified)? Is this information clear and unambiguous?

> Plausibility checks (for instance in a business or MIS application): the CRUDL check relates to the expectation that for each object of interest for which data is stored persistently there will be at least one functional process that Creates (i.e. stores persistently) the data describing the object, one that Reads the data, one (often more than one) that Updates the persistently stored data, one that Deletes the data and one that Lists all instances of the object of interest.

> Cross-checks. Is each functional process identified reflected in the software artifact (i.e. is the software artifact complete)?

A.2 Checks on the quality of measurements

A.2.1 Checklist for the Measurement Strategy Phase

> Is it stated why the measurement is required, what the result will be used for and what is the target accuracy?

> Is the purpose to measure all the delivered software or just the newly developed or enhanced software?

> Is the overall scope stated? Are the scopes stated of the individual pieces of software to be measured separately?

> If within the overall scope the components of a piece of software are distinguished, are their levels of decomposition also identified?

> Does the overall scope include software in more than one layer? If so, is the scope of any one measurement limited to one layer?

> For each piece of software to be measured, are all its functional users identified? Are the functional users in agreement with the purpose of the measurement?

> If the overall scope of the measurement contains several software artifacts, have their levels of granularity been identified? Are the measurements made at the same level of granularity?

> If a measurement was scaled to the level of granularity of functional processes or an approximate sizing method was used, has the accuracy of the resulting size been estimated?

A.2.2 Checklist for the Mapping Phase

> Is each functional process associated with a triggering event and a functional user? Does each identified event trigger at least one functional process?

> Does each functional process represent a view of the software of the functional user that triggered it?

> Does every functional user either detect at least one event, and/or receive data from a functional process?

> Is each functional process indivisible, i.e. there is no part of the functional process that can be triggered by a separate event?

> Is each functional process complete (i.e. does it include all that is required to be done in response to the triggering event)?

> Are the objects of interest identified? Is each object of interest identified really an object of interest to a functional user?

> Is each attribute in the data movements related to exactly one object of interest?

> For each functional process, is it clear which data it has to store persistently or retrieve from persistent storage itself and for which data it must call other software to store or retrieve it?

A.2.3 Checklist for the Measurement Phase

> If a functional process must store data persistently or retrieve data from persistent storage, are one or more Write or Read data movements identified respectively?

> If a functional process must communicate with another piece of software (i.e. a functional user) to store or retrieve data, is one Exit per object of interest identified for the request and one Entry per object of interest identified for the returned data or message?

> Does the data stored persistently for each object of interest get Read by a functional process of the software being measured or by some other software?

> If a functional process must obtain data from a functional user but the latter does not need to be told what data to send, is one Entry and no Exit identified?

> If the FUR require a functional process to output messages without user data (e.g. error messages), is a single Exit identified?

> If data attributes of one object of interest must be entered into a functional process, is one Entry identified (unless the FUR specify otherwise)? In like manner for Read, Write or Exit data movement?

> (In the business application domain only) Are control commands ignored (e.g. navigation commands, page up/down, clicking ‘OK’ for confirmation, etc)?

> Have clock and timing triggering events been identified?

> For an enhancement project, has the analysis identified all data movements that have been added, changed or deleted?

A.3 Checks on areas of commonly-made errors

> Identify assumptions and interpretations, check their validity and check the relevant parts of the measurement, especially with regard to:

- the data model, e.g. an entity-relationship model, a relational data model or a model of object classes
- data that are not attributes of an object (that is normally) of interest, e.g. standard screen headings, data on a report such as page numbers
- parameter tables
- drop-down lists
- objects of interest sub-types
- multiple occurrences of the same type of data movement
- recursive relationships between objects of interest
- re-use of functionality
- omitted 'list before update' (i.e. measuring physical not logical screens)
- logon functionality
- drilldown functionality
- the confirmation/error messages Exit
- clock and timing triggering events

> (For functional processes incorrectly combined into one functional process) Identify missing functional processes by verifying that any triggering event corresponds with a functional process

> (For functional processes incorrectly split up into functional processes) Identify redundant functional processes by verifying that any functional process corresponds with one triggering event.