Article category: SCOPE

Software development:
Function point analysis

By Ad Sparrius

> *Progress in all scientific and engineering fields has been closely coupled to accurate measurements of basic phenomena. Without the ability to measure voltage, current and impedance, there could be no electrical engineering. Software tries to be the only exception. When the Danish astronomer Ole Rømer first attempted to measure the speed of light, his results only indicated 225 000 km/s which differs from today's value of 299 793 km/s by 25%. However, prior to Rømer's publication in 1676 most scientists thought that the speed of light was infinite. Even Rømer's incorrect result started scientists along an extremely useful path. - Capers Jones, Applied Software Measurement – Assuring Productivity and Quality*

This article focuses on estimation issues associated with software development.

Software estimation

The two major software estimation problems are –

- estimate software size and complexity, and
- estimate software productivity.

Once these size and productivity estimates have been obtained, a cost and duration estimate can be determined.

The natural units of software production are lines of code, but the natural units of software consumption are functions. *Function points* represent by far the best independent variable for software size and complexity estimates. Function points are technology independent, consistent and repeatable. A general ledger application of 850 function points will remain that size whether written in Cobol or in C++, and irrespective of whether object-orientation or procedural techniques were used, or whether or not graphic user interfaces were used. A function point is not the same as a user function to be implemented.

A function point count is based on the user's requirements – how many inputs are required? How many outputs? How many calculations are required? How many database queries will be required? Each of the counts for these factors is weighted, the weighted counts are summed, and the total is adjusted for complexity to become the function point count. Function points are only based on functional requirements and can thus be estimated and counted early in the software development life cycle. The counting variance between trained counters is about ±10%, in other words repeatability is good [1].

Microsoft Excel and Word each consist of about 5 000 function points, and Windows 95

Ad Sparrius

Ad Sparrius graduated in electronic engineering from Stellenbosch and Berkeley, California, and was awarded an MBL by UNISA.

He has extensive project management experience in the high-technology industry. He consults in engineering management to a broad cross section of South African industry. Many people have attended his courses or guest lectures.

He is a professor at UNISA's Graduate School of Business Leadership.

Tel 082 574-0266
Fax (012) 361-3828
e-mail:
ad_sparr@iafrica.com

and Windows NT each contains about 50 000 function points [2].

Productivity

Function points merely measure the size and complexity of software, not the effort involved in developing it. Effort estimates require productivity factors. For instance, team A develops 100 functions points in 10 personmonths but team B develops the same application in 5 personmonths. The productivity of team A is 10 FP/personmonth, but team B's productivity is 20 FP/personmonth. The overall average productivity for management information systems is –

1 000 FP	4.63 FP/pm
10 000 FP	2.98 FP/pm

For most projects the peak of the productivity distribution, also known as the mode, lies between 5 and 15 FP/personmonth. The average management information system application consists of 250 function points. The average productivity for Cobol is 8-10 FP/personmonth, and for a client-server application using object-oriented design about 16 FP/personmonth.

The productivity of software engineers can exhibit a range from the most to the least productive of 25 to one. Quality in terms of faults per function point can vary by as much as ten to one.

Two productivity distributions [2] are shown in Table 1.

Productivity (function points/person-month)	***Productivity distribution (%) for all software domains, at 1000 function points***	***Productivity distribution (%) for management information systems***
≥ 100	0	0
75-100	1	4
50-75	2	7
25-50	3	12
15-25	20	18
5-15	55	35
1-5	15	18
< 1	4	5
Total	*100*	*100*

Table 1: Productivity distributions

The most likely productivity range is 5-15 FP/personmonth, with an average of 10 FP/personmonth. Higher productivities are doubtful [2].

In real life there are projects that have a productivity of 20 FP/personmonth, and even 100 FP/personmonth. However, any project larger than 1 000 function points that reports a productivity rate higher than 10 FP/personmonth needs to be carefully validated before that productivity data can be accepted as legitimate.

Once a function point count has been obtained, many estimating relationships based on comparative methods may be used. Some examples are listed below.

Schedule

The schedule from the start of requirements analysis to delivery to the first paying customer is given by:

$$\text{Schedule} = (\text{FP})^{0.4} \text{ calendar months}$$

Surprisingly, both the start point and the end point of software development are notoriously difficult to determine.

Effort

Effort estimates include *all* activities, of which design, coding and testing is merely a part. Where will this software development effort be spent? The software development effort for all

software domains at a size of 1 000 function points is shown in Table 2. Many development activities have nothing to do with coding, for instance paper work!

Defect potential

A "defect" refers to the total of possible errors in software from five separate sources:

Activity	*Effort (%)*
Management	15
Paperwork	25
Defect removal	30
Coding	30
Total	*100*

Table 2: Software development effort

- errors in requirements,
- errors in design,
- errors in source code,
- errors in user documentation, and
- errors associated with poor fixes or secondary errors introduced while fixing a primary error.

Total number of potential defects = $(FP)^{1.25}$

The origin of these defects will typically be distributed as shown in

Origin of defect	*Occurrence (%)*
Requirements	15
Design	30
Coding	35
User documentation	10
Poor fixes	10
Total	*100*

Table 3: Origin of defects

Defect removal

The methods of defect removal include formal design reviews and code inspections, as well as various tests. As a rule of thumb, any given defect removal method will find about 30 percent of the defects present. Defect removal efficiency is first calculated on the anniversary of software release.

If a cumulative defect removal efficiency of 95% is desired, how many defect removal stages will be needed?

Defect removal stages = $(FP)^{0.3}$

This result shows why software development normally needs multi-stage design reviews, code inspections and various levels of testing from unit test through system testing. Incidentally, the major cost and schedule drivers of software development are those activities associated with defect removal.

The client side and the server side of client-server applications may have different productivity rates and may even use different tools and languages. Client-server applications typically have about 20 percent more defects than traditional applications, and remove about 10 percent fewer of these defects before deployment [2].

Test cases

The total number of test cases which will be needed will be about –

Total test cases = $(FP)^{1.2}$

Each test case will be executed about four times during development.

A test case is a set of test inputs, execution conditions and expected results, which are developed for a particular objective, such as to exercise a specific program path or to verify compliance with a specification requirement. Test cases are samples from the space of all possible execution sequences.

Consider the development of a square root function. One set of test cases would use valid inputs and specify the correct results, for instance the square root of 400 is 20. Another set of test cases would use invalid inputs and specify the appropriate error message, for instance the square root of – 400 would require an error message "Invalid input value". Test cases form the backbone of the entire software test process, and the total number of test cases determines the duration of the test process. The more complex the software the more test cases will be needed and the longer the development process.

For Microsoft Word having 5 000 function points, the number of potential defects would be $(5\ 000)^{1.25}$ = 42 000, about $(5\ 000)^{0.3}$ = 13 defect removal stages would be needed for a defect removal efficiency of 95%, and about $(5\ 000)^{1.2}$ = 27 500 test cases would be needed.

Requirements creep

User requirements will creep at an average rate of 1 percent per month

over the entire development period.

Life expectancy

The life expectancy of software will be –

$$\text{Life expectancy} = (\text{FP})^{0.25} \text{ years}$$

For instance, the life expectancy of Microsoft Word having 5000 function points would be 8.4 years. After that period the software would no longer be a useful product and would have to be modified or replaced. Of course some users would continue using the software much longer than this – you would be surprised at how many people still use Wordstar, one of the first word processors! Of course competitive realities in a consumer market might demand ongoing software improvement, meaning that its commercial life could be only a few months, but such issues are not included in this life expectancy estimate.

The life expectancy would be quite reasonable for software that is not commercially available. Incidentally, software life expectancy is similar to the life expectancy of animals — heavier animals live longer. Elephants generally live longer than rats.

Assignment scope

The assignment scope of a software maintainer ranges from 500 to 2 000 function points. In other words, the number of software maintainers N needed is –

$$\text{FP}/2\,000 \leq \text{N} \leq \text{FP}/500$$

The assignment scope for new developments is 50-100 function points per developer.

For instance, Microsoft Word having 5000 function points would require between 50 and 100 developers, and between 2 and 10 maintainers.

Paper deliverables

Software is a paper-intensive industry! The total number of pages that will be created in requirements, specifications, project plans, user manuals, etc. is given by –

$$\text{Total page count} = (\text{FP})^{1.15}$$

Each page contains about 400 English words.

Microsoft Word having 5000 function points would generate about $(5000)^{1.15}$ = 18 000 pages of paper.

Backfiring

Backfiring assumes a strong correlation between the needed functionality in terms of function points and the implemented lines of source code, excluding comments. If lines of code can be counted, then the function points which it implements can be determined by means of backfiring. It is now easy to calculate productivity as the software development project progresses from version to version. The margin of error for backfiring ranges from 25% to more than 200%. Typical backfiring factors are shown in Table 4.

Example
It has been reported that Microsoft's Windows 2000 operating system consists of 29 million lines of C++ code. This includes the kernel, Internet browser, transaction processor and all the device drivers (a total of 8 million lines of code for 1 200 different video cards).

Backfiring results in 207k ≤ FP ≤ 725k, with a modal value of 527k FP. Testing and debugging consumed 90-95% of all effort. The development team consisted of 4 200 people (2 000 from Microsoft itself, 800 from its partners for instance Intel, and 1 400 contractors), plus another 1 500 people employed elsewhere, for instance in India.

Historical database

The database for this comparative analysis is 6 753 projects, of which 465 were management information system projects of size 1 000 to 10 000 function points. Inevitably the database contains data from completed projects – three quarters of the data is from the 1980's and later.

Software development practice is improving all the time, but it will be some time before the influence of any one such practice is evident in the database.

Development duration

The cost-optimum development duration to first shipment is [3] –

$$T = 2.5 (pm)^{1/3}$$

months where pm is the personmonths of effort.

As the planned schedule gets longer than the optimum duration, the cost increases slowly, but if the planned schedule is shorter than optimum, the cost rises sharply. Hardly any project succeeds in less than ¾ of the optimum duration, regardless of the number of people used!

References

1. Jones; *Applied Software Measurement – Assuring Productivity and Quality*, second edition, McGraw-Hill, 1997.
2. Jones; *Software estimating rules of thumb*, IEEE Computer, vol 29, no 3, March 1996, pp 116-118.
3. Boehm; *Software Engineering Economics*, Prentice Hall, 1991. www.ifpug.org
4. International Function Point User Group; *Function Point Counting Practices Manual*, release 4.1, May 1999.

Language	*Source Lines of Code/ Function point*		
	minimum	*mode*	*maximum*
Cobol	65	107	150
C++ default	40	55	140
C default value	60	128	170
ANSI Basic	35	64	100
Pascal	50	91	125
Oracle, IDMS, Sybase		40	
Object-oriented default	13	29	40
Assembler	237	320	416
Spreadsheet	3	6	9
Natural language	1 800	3 200	5 000

Table 4: Backfiring

Hardly any project succeeds in less than ¾ of the optimum duration, regardless of the number of people used!

Microsoft's Windows 2000 operating system: Testing and debugging consumed 90 –95% of all effort. The development team consisted of 4 200 people

---

This article was first published in ProjectPro magazine Vol 10 No 6, November 2000

Back to the top