EARLY SIZING AND ESTIMATING OF SOFTWARE QUALITY, SCHEDULES, EFFORT, AND COST

Capers Jones, Vice President and Chief Technology Officer; Namcook Analytics LLC,

Abstract

Most methods of software sizing and estimating are based on software requirements and design documents, or on the source code itself. For both new applications and enhancements this means that substantial funds will have been expended before sizing and estimating take place. Early sizing and estimating pay handsome dividends due to better results for on-schedule and within-budget projects, as shown by table 1 from the paper:

Table 1: Results of Early Sizing and Estimating
(Assumes 1000 Function Points and Java code)

	On-Time Delivery	In budget Delivery	Defect Removal Efficiency	$ per Function Point
Automated sizing and estimating before requirements	95.00%	95.00%	97.50%	$950
Automated sizing and estimating after requirements	80.00%	87.00%	92.50%	$1,100
Manual sizing and estimating after requirements	55.00%	60.00%	88.00%	$1,350
No formal sizing or estimating	44.00%	38.00%	85.00%	$1,800

Namcook Analytics has developed a new method of early software sizing and estimating based on pattern matching that can be used prior to requirements. This new method permits very early estimates and very early risk analysis before substantial investments are made.

INTRODUCTION

Software sizing and estimating have been weak areas of software engineering since the industry began. Even in 2017 a majority of the world's software projects use educated guesses for sizing and inaccurate and optimistic manual estimates instead of accurate automated parametric estimates. Large companies are more sophisticated than small companies and tend to have more sophisticated sizing and estimating methods and tools. It is useful to look at sizing and estimating methods since the software industry began:

1950 to 1959

Sizing was based on lines of code (usually physical). Estimating was a combination of guesswork and sparse historical data. In this era assembly was the only language at the beginning of the decade so lines of code (LOC) still worked.

1960 to 1969

Sizing was based on lines of code. Estimating was still based on guesswork but some companies such as IBM collected useful historical data that were helpful in estimation. Projects grew larger so estimating requirements and design became important. Languages such as COBOL, Algol, and others were developed and LOC metrics started to encounter problems such as difficulty in counting lines of code for projects with multiple languages.

1970 to 1979

By the end of this decade over 80 programming languages were in existence and LOC metrics were proven to be inaccurate and unreliable by IBM. LOC metrics can't measure requirements and design, which by 1979 were more expensive than code itself. Also, LOC metrics penalize high-level languages such as APL and PL/I.

IBM was the first company to perform a formal study of the errors and problems with LOC metrics, which occurred about 1972 with the author as the primary IBM researcher. This study and the proof of LOC failure led IBM to fund the development of function point metrics.

Due to major problems and proven errors with lines of code metrics, IBM created the function point metric for sizing and estimating software projects.

In 1973 the author developed IBM's first parametric estimation tool, which was coded in APL by Dr. Charles Turk. Also during this decade other estimating pioneers such as Dr. Barry Boehm, Dr. Howard Rubin, Frank Freiman, and Dr. Larry Putnam developed early parametric estimation tools.

Function points were put into the public domain by IBM in 1979 and expanded all over the world. By the end of this decade many leading organizations such as IBM, ITT, TRW, RCA, and the Air Force used proprietary parametric estimation tools and also used function point metrics. The technique of "backfiring" or mathematical conversion from source code to function points was developed by IBM in this decade and began to be used for sizing legacy applications.

1980 to 1989

During this decade function points became a major global metric for sizing and estimating. The author's SPQR/20 estimation tool released in 1984 was the world's first parametric estimation tool based on function points. It was also the first that included sizing for all deliverables (requirements, design, code, test cases, etc.). The inventor of function point metrics, A.J. Albrecht, had retired from IBM and came to work for the author.

Only function point metrics support sizing of requirements, design, user documents, and test cases; this cannot be done using lines of code metrics. Other parametric estimating tools such as CheckPoint, COCOMO, Estimacs, KnowledgePlan, SEER, SLIM etc. also entered the commercial market.

Most major companies used either commercial or proprietary parametric estimation by the end of this decade. (The author designed proprietary estimation tools under contract for several telecommunication companies and also taught software sizing and estimation at AT&T, Bell Northern, GTE, ITT, Motorola, Nippon Telephone, Nokia, Pacific Bell, Siemens, Sprint, and others.)

The International Function Point User's Group (IFPUG) was created in Canada and began to provide certification examinations to ensure accurate function point counts. IFPUG later moved its headquarters to the United States.

1990 to 1999

Due to the initial success and value of IFPUG function points, this decade saw the creation of a number of function point metric "clones" that differed somewhat in their counting rules. Some of these functional metric clones include in alphabetical order: COSMIC function points, Engineering function points, Fast function points, Feature points, FISMA function points, NESMA function points, Story points, Unadjusted function points, and Use-case points. Most of these use counting rules similar to IFPUG but have various additional rules or changed rules. This causes more confusion than value. It also introduced a need for conversion rules between the various functional metrics, such as the conversion rules built into Software Risk Master (SRM).

The International Software Benchmark Standards Group (ISBSG) was created in 1997 and began to provide software benchmark data using only function points. (Lines-of-code data is hazardous and unreliable for benchmarks.)

Other benchmark providers also began to offer benchmarks such as the author's former company, Software Productivity Research (SPR). Two former vice presidents at SPR also formed software benchmark companies after SPR was sold in 1998.

Although function point metrics were widely used and were accurate when counted by certified personnel, they were also slow and expensive. An average certified function point counter can count about 500 function points per day, which limited the use of function points to small projects.

By the end of this decade most U.S. telecom companies employed or contracted with between 5 and 12 certified function point counters.

2000 to 2010

Function point metrics had become the global basis of software benchmarks and were widely used in parametric estimation tools. However the slow counting speeds and high costs of manual function point analysis led to research in easier and faster methods of function point analysis.

Many companies developed function point tools that provided mathematical support for the calculations. These speeded up function point counting from about 500 function points per day to more than 1000 function points per day.

Several companies (CAST Software and Relativity Technologies) developed automated function point counting tools. These tools examined source code from legacy applications and created function points based on this code analysis. These tools only work for applications where code exists, but instead of 500 function points per day they can top 50,000 function points per day.

In 2012 IFPUG issued the new SNAP metric (software non-functional assessment process. This metric has been added to the Namcook sizing method.

The author and Namcook Analytics developed a proprietary sizing method based on pattern matching that is part of the Software Risk Master (SRM) estimating tool.. This method uses a formal taxonomy and then extracts size data from the Namcook knowledge base for projects that have the same taxonomy pattern.

The Namcook sizing method is included in the Software Risk Master (SRM) estimating tool. SRM sizing is unique in being able to size and estimate projects before requirements, or 30 to 180 days earlier than any other known method.

The SRM method is the fastest available method of sizing, and averages about 1.8 minutes to size any application. Of course measured speed varies with the sizes of the applications themselves, but SRM can size at speeds of well over 300,000 function points per day.

SRM sizing is also unique in being able to predict size in a total of 23 metrics at the same time: all forms of function points, logical and physical lines of code, story points, use-case points, and even RICE objects for ERP packages.

SRM also collects new benchmark data every month, because estimating tools need continuous updates to their knowledge bases to stay current with technical changes such as cloud computing, SaaS, estimating social network software, and other technology innovations. In fact the commercial estimating companies all collect benchmark data for the same reason.

The State of the Art of Sizing and Estimating in 2017

As of 2017 the state of the art varies with the size and sophistication of the company. Large technology companies such as medical devices, computers, avionics, telecommunications etc. typically use multiple sizing and estimating methods and look for convergence. Most employ or use function point counters. Most technology companies use high-end commercial estimation tools such as Software Risk Master (SRM) or have built their own proprietary estimation tools. Some smaller companies and universities use the open-source COCOMO estimating tool, which is available without cost.

Mid-sized companies and companies in the banking, insurance, and other areas are not quite as sophisticated as the large technology companies. (It is interesting that the Bank of Montreal was the first major company to use function points and was a founder of IFPUG.)

However a recent survey of over 100 companies found that function point metrics were now dominant for estimating and benchmarks in the U.S., Europe, Japan, and Brazil and these countries have many more function point users than other metrics such as story points or the older LOC metrics. About 70% of mid-sized companies still use manual estimates but about 30% use one or more parametric estimating tools such as Software Risk Master (SRM).

Many agile projects use manual estimates combined with the "story point metric." Unfortunately story points have no ISO or OMG standards and vary by hundreds of percent from company to company. They are almost useless for benchmarks due to the low quantity of available data and the poor accuracy of story points for either estimates or measurements.

In 2012 a new metric for non-functional requirements called "SNAP" was created by IFPUG and is now starting to be used. However as of 2016 this metric is so new that not a great deal of data exists, nor do all companies use it. This metric needs additional definitions and continued analysis.

Small companies with less than 100 employees only build small applications where risks are low. About 90% of these companies use manual estimates. Most are too small to afford function point consultants and too small to afford commercial estimating tools so they tend to use backfiring and convert code size into function points. They still need function points because all reliable benchmarks are based on function point metrics. Some small companies use COCOMO because it is free, even though it was originally calibrated for defense software. Table 1 shows the economic advantages of using automated sizing and estimating tools such as Software Risk Master (SRM).

Table 1: Results of Early Sizing and Estimating
(Assumes 1000 Function Points and Java code)

	On-Time Delivery	In budget Delivery	Defect Removal Efficiency	$ per Function Point
Automated sizing and estimating before requirements	95.00%	95.00%	97.50%	$950
Automated sizing and estimating after requirements	80.00%	87.00%	92.50%	$1,100
Manual sizing and estimating after requirements	55.00%	60.00%	88.00%	$1,350
No formal sizing or estimating	44.00%	38.00%	85.00%	$1,800

As can be seen early sizing and estimating using a tool such as Software Risk Master (SRM) can lead to much better on-time and within-budget performance than older manual estimating methods or delayed estimates after requirements are completed.

Hazards of Older Metrics

Even some users of function point metrics are not fully aware of the problems with older metrics. Here are short summaries of metric hazards in alphabetical order:

Automated function points: Several companies such as CAST Software and Relativity Technologies have marketed automated function point tools that derive function point totals from an analysis of source code. These tools have no published accuracy data. They also can only be used on legacy software and cannot be used for early sizing and estimating of new software applications before code exists.

Cost per defect penalizes quality and is cheapest for the buggiest software. This phenomenon was discovered circa 1972 by the author and colleagues at IBM. Cost per defect cannot be used at all for zero-defect software. The cost per defect for software with 1000 released defects will be much cheaper than the same software with only 10 defects. Cost per defect is useless for zero-defect software, which should be the goal of all projects. Defect removal costs per function point provide a much better basis for studying software quality economics than cost per defect.

Design, code, and unit test (DCUT) metrics are embarrassingly bad. The sum total of effort for design, code, and unit test is less than 37% of total software effort. Using DCUT is like measuring only the costs of the foundations and framing of a house, and ignoring the walls, roof, electrical systems, plumbing, etc. Only the software industry would use such as poor metric as DCUT. All projects should measure every activity: business analysis, requirements, architecture, design, documentation, quality assurance, management, etc.

Lines of code (LOC) metrics cannot measure non-coding work such as requirements, architecture, design, and documentation which are more expensive than the code itself. (Coding on large systems may only comprise 30% of total costs.) LOC cannot measure bugs in requirements and design, which often are more numerous than coding bugs. Even worse, LOC metrics penalize high-level languages and make low-level languages such as assembly and C look better than high-level languages such as Visual Basic and Ruby. Also, many languages use buttons or controls and allow "programming" without even using lines of code. LOC has no ISO standard counting rules (physical and logical code are counted about equally), and also no certification exams. There are automatic counting tools for LOC but these vary in what they count. Finally, an average application in 2015 uses at least two languages and sometimes up to a dozen different programming languages. Code counting for multiple programming languages is very complex and slow. Typical combinations are Java, HTML, MySQL, and possibly others as well.

Technical debt by Ward Cunningham is a brilliant metaphor but not yet an effective metric. Technical debt has no ISO standards and no certification exams. Among the author's clients technical debt varies by more than 200% between companies and projects. Worse, technical debt only covers about 17% of the total costs of poor quality. Missing with technical are canceled projects that are never delivered; consequential damages to users; litigation costs for poor quality; and court awards to plaintiffs for damages caused by poor quality.

Story point metrics are widely used with agile projects. However story points have no ISO standards or OMG standards and no certification exams. Among the author's clients story points vary by more than 400% between companies and projects. There are few if any benchmarks based on story points.

Use-case metrics are widely used with RUP projects. However use-case points have no ISO standards and no certification exams. Among the author's clients use-case points vary by more than 100% between companies and projects. There are few if any benchmarks based on use-case points.

Overall function point metrics provide the most stable and effective metrics for analyzing software quality economics, software productivity, and software value. The major forms of function points have ISO standards and certification exams; unlike the older and hazardous metrics discussed above.

As illustrated elsewhere in this report the detailed metrics used with function points include but are not limited to:

Table 2: Software Risk Master (SRM) Function Point and SNAP Usage Circa 2017

1. Predicting size in function points, SNAP, LOC and a total of 23 metrics
2. Early sizing and risk analysis via pattern matching before full requirements
3. Sizing of internal, COTS, and open-source applications
4. Sizing and estimating both new projects and legacy repairs and renovations
5. Sizing and estimating 15 types of software (web, IT, embedded, defense, etc.)
6. Source code sizing for 84 programming languages and combinations of languages
7. Sizing requirements creep during development (> 1% per calendar month)
8. Sizing post-release requirements growth for up to 10 years (> 8% per year)
9. Sizing defect potentials per function point/SNAP point (requirements, design, code, etc.)
10. Defect prevention efficiency (DPE) for JAD, QFD, modeling, reuse, etc.
11. Defect removal efficiency (DRE) for pre-test and test defect removal methods.
12. Document sizing for 30 document types (requirements, design, architecture, etc.)
13. Sizing test cases per function point and per SNAP point for all forms of testing
14. Estimating delivered defects per function point and per SNAP point
15. Activity-based costs for development
16. Activity-based costs for user effort on internal projects
17. Activity based costs for maintenance
18. Activity-based costs for customer support
19. Activity-based costs for enhancements
20. Occupation-group effort for 25 common software skills (coders, testers, analysts, etc.)
21. Total cost of ownership (TCO) including cyber-attack costs
22. Cost of quality (COQ) for software applications including cyber-attacks and litigation
23. Estimating the newer technical debt metric which is ambiguous in 2016
24. Risk probabilities for 30 common software risk factors (delays, overruns, cancellation)
25. Estimating productivity and quality results for 60 software development methodologies
26. Estimating ERP deployment, customization, and training costs
27. Software litigation costs for failing outsource projects (both plaintiff and defendant)
28. Estimating venture funding rounds, investment, equity dilution for software startups
29. Estimating cyber-attack deterrence and recovery costs (new in 2016)
30. Portfolio sizing for corporate portfolios (> 5000 applications,10,000,000 function points, and 1,500,000 SNAP points) including internal, COTS, and open-source.

All 30 of these sizing features are included in the Software Risk Master (SRM) sizing methodology as of 2017.

Metrics Used with Function Point Analysis

The counting rules for function points are available from the various function point associations and are too complicated to discuss here. If a company wants to learn function point counting, the best methods are to either hire certified function point counters or send internal personnel to learn function point analysis and take a certification exam offered by the function point associations.

The current IFPUG counting rule manual is available from the IFPUG organization and is about 125 pages in size: too big to summarize here. Counting rules are also available from other function point communities such as COSMIC, FISMA, NESMA, etc.

Once the function point total for an application is known, then function points can be used with a variety of useful supplemental metrics to examine productivity, quality, costs, etc. Some of the leading metrics used with function points include in alphabetical order:

Assignment scope

This is the amount of work typically assigned to a software team member. It can be expressed using function points or natural metrics such as pages of documents and lines of code. For example a technical writer might be assigned a user manual of 200 pages. Since software user manuals average about 0.15 pages per function point that would be an assignment scope of 30 function points.

Typical assignment scopes using function points for a project of 1000 function points would be:

Requirements = 460 function points

Design = 345 function points

Coding = 130 function points

Testing = 150 function points

This kind of data is available for 40 activities from Namcook Analytics LLC. This data is a standard feature of Software Risk Master (SRM) but limited to 7 activities.

Cost per function point

As of 2017 cost per function point is one of the most widely used economic metrics in the world. Several national governments such as Brazil and South Korea demand cost per function point in all bids and software contracts. India uses cost per function point to attract business to Indian outsource companies. The cost per function point metric can be used for full projects and also for individual activities such as requirements, design, coding, etc.

There are several cautions about this metric however. For long-range projects that may take more than 5 years inflation needs to be factored in. For international projects that may include multiple countries local costs and currency conversions need to be factored in. In the U.S. as of 2015 development costs per function point range from less than $500 for small internal projects to more than $3,000 for large defense projects.

Cost per function point varies from project to project. Assuming a cost structure of $10,000 per month and 1000 function points typical costs per function point would be:

Requirements = $41.79

Design = $66.87

Coding = $393.89

Testing = $236.34

Here too these are standard results from Software Risk Master (SRM). This kind of data is available for 40 activities from Namcook Analytics LLC. SRM shows 7 activities.

Defect potentials

Defect potentials are the sum total of bugs that are likely to be found in requirements, architecture, design, code user documents, and bad fixes or secondary bugs in bug repairs. U.S. totals for defect potentials range from < 2.00 defects per function point to > 6.00 defects per function point. This metric is also used for specific defect categories. Requirements defects per function point range from <0.25 per function point to > 1.15 per function point. The full set of defect potentials include defects in requirements, architecture, design, code, documents, and "bad fixes" or secondary bugs in defect repairs themselves. There are also defects in test cases, but these are very seldom studied so there is not enough available data to include test-case defects in defect potentials as of 2016.

Defect potentials are ONLY possible with function point metrics because LOC metrics cannot be used for requirements and design defects. Typical values for defect potentials in function points circa 2017 are shown below:

Table 3: Average Software Defect Potentials circa 2016 for the United States

- Requirements 0.70 defects per function point
- Architecture 0.10 defects per function point
- Design 0.95 defects per function point
- Code 1.15 defects per function point
- Security code flaws 0.25 defects per function point

- Documents 0.45 defects per function point
- Bad fixes 0.65 defects per function point
- **Totals** **4.25 defects per function point**

As can be seen, defect potentials include bugs in many sources and not just code. As can be seen, requirements, architecture, and design defects outnumber code defects. Defect potential estimation is a standard feature for Software Risk Master (SRM).

Defect removal efficiency (DRE)

This metric does not use function points themselves, but rather shows the percentage of defect potentials removed before release. Typical values would be 80% of requirements defects are removed before release but 98% of code defects. Software Risk Master (SRM) predicts both defect potentials and individual removal efficiency levels for requirements defects, architecture defects, code defects, document defects, and bad-fix injections.

Typical values for defect removal efficiency are about the following:

Requirements defects = 75%

Design defects = 85%

Architecture defects = 90%

Code defects = 97%

Defect removal efficiency is a standard feature of Software Risk Master (SRM). DRE in SRM includes 1 pre-test inspections, 2 static analysis, 3 desk checking, and 4 pair programming.

Test DRE is shown for six kinds of testing: 1 Unit, 2 Regression, 3 Component; 4 Performance, 5 System, Acceptance.

Function points per month

This is a common productivity metric but one that needs to be adjusted for countries, industries, and companies. Work-hours-per-function-point is more stable from country to country. The typical number of work hours in the U.S. is 132 hours per month; in India it is about 190 hours per month; in Germany it is about 116 hours per month. Thus the same number of work hours would have different values for function points per month. Assume a small project took exactly 500 work hours. For India this project would take 2.63 months; for the U.S. 3.78 months; for Germany 4.31 months. The metric of work hours per function point is stable across all countries, but function points per month (and the older LOC per month) vary widely from country to country.

Production rate

This metric is the amount of work a software team member can perform in a given time period such as an hour, day, week, or month. This metric can be expressed using function points or natural metrics such as Lines of code or pages. For example a technical writer might be able to write 50 pages per month. A programmer may be able to code 1,000 lines of code per month. A tester may be able to run 500 test cases per month, and so on. The same activities can also be measured using work hours per function point, or a combination of function points and natural metrics.

Requirements creep

Because applications add new requirements and new features during development, size must be adjusted from time to time. Requirements grow and change at measured rates of between 1% per calendar month and about 4% per calendar month. Thus an application sized at 1,000 function points at the end of requirements may grow to 1,100 function points by delivery. Software keeps growing after release, and the same application may grow to 1,500 function points after three or four years of use. Software Risk Master (SRM) predicts growth and can also measure it. (This is not a feature of most parametric estimation tools.)

Work hours per function point

This is a very common metric for software productivity. It has the advantages of being the same in every country and also of being useful with every software development activity. Software Risk Master (SRM) uses this as a standard metric for all estimates as shown below:

1. Requirements = 0.60 work hours per function point
2. Design = 0.90 work hours per function point
3. Coding = 5.00 work hours per function point
4. Testing = 3.50 work hours per function point
5. Quality = 0.50 work hours per function point

6. Documents = 0.40 work hours per function point
7. Management = 2.00 work hours per function point

TOTAL = 12.90 work hours per function point

Note: these values are just examples and not intended for use in actual estimates. There are wide ranges for every activity. Also, the example only shows 7 activities, but similar data is available from Namcook Analytics LLC for 40 activities.

The same metric or work hours per function point can also be used to measure user costs for internal user effort, training costs for customers and team members, and even marketing and sales effort for commercial software packages. It can also be used for customer support, bug repairs, and even project management.

Function points are a powerful and useful metric but need additional metrics in order to actually estimate and measure real projects.

Application Sizing Using Pattern Matching

The unique Namcook pattern matching approach is based on the same methodology as the well-known Trulia and Zillow data bases for real-estate costs.

With the real-estate data bases home buyers can find the costs, taxes, and other information for all listed homes in all U.S. cities. They can specify "patterns" for searching such as size, lot size, number of rooms, etc.

Following are the main topics used for software pattern matching in the Namcook Software Risk Master (SRM) tool:

Table 4: Patterns for Application Sizing and Risk Analysis

1. Local average team salary and burden rates
2. Paid and unpaid overtime planned for projects
3. Planned start date for the project
4. Desired delivery date for the project
5. Country or countries where the software will be built
6. Industry for which the software is intended
7. Locations where the software will be built (states, cities)
8. Experience levels for clients, team, management
9. Development methodologies that will be used (Agile, RUP, TSP, etc.) *
10. CMMI level of the development group *
11. Programming language(s) that will be used (C#, C++, Java, SQL, etc.) *

12. Amount of reusable materials available (design, code, tests etc.) *
13. Nature of the project (new, enhancement, etc.) *
14. Scope of the project (subprogram, program, departmental system, etc.) *
15. Class of the project (internal use, open-source, commercial, etc.) *
16. Type of the project (embedded, web application, client-server, etc.) *
17. Problem complexity ranging from very low to very high *
18. Code complexity ranging from very low to very high *
19. Data complexity ranging from very low to very high *
20. Number of anticipated users (for maintenance estimates)

Note: Asterisks "*" indicate factors used for pattern analysis for sizing.

All of these topics are usually known well before requirements. All of the questions are multiple choice questions except for start date and compensation and burden rates. Default cost values are provided for situations where such cost information is not known or is proprietary. This might occur if multiple contractors are bidding on a project and they all have different cost structures.

The answers to the multiple-choice questions form a "pattern" that is then compared against a Namcook knowledge base of more than 25,000 software projects. As with the real-estate data bases, software projects that have identical patterns usually have about the same size and similar results in terms of schedules, staffing, risks, and effort.

Sizing via pattern matching can be used prior to requirements and therefore perhaps six months earlier than most other sizing methods. The method is also very quick and usually takes less than 5 minutes per project. With experience, the time required can drop down to less than 2 minutes per project.

The pattern matching approach is very useful for large applications > 10,000 function points where manual sizing might take weeks or even months. With pattern matching the actual size of the application does not affect the speed of the result and even massive applications in excess of 100,000 function points can be sized in a few minutes or less.

This method of sizing by pattern matching is covered by a U.S. utility patent application submitted to the Patent Office in January of 2012. The algorithms for sizing by pattern matching are included in the author's tool Software Risk Master™ (SRM).

The method of sizing by pattern matching is metric neutral and does not depend upon any specific metric. However due to the fact that a majority of the author's clients use function point metrics as defined by the International Function Point Users Group (IFPUG) the primary metric supported is that of IFPUG function points counting rules 4.2. There are of course more projects measured using IFPUG function points than those available using other metrics.

Many additional metrics can also be based on sizing via SRM pattern matching including but not limited to:

Table 5: Metrics Supported by Namcook Pattern Matching

1. IFPUG function points
2. Automated code-based
3. Automated UML-based
4. Backfired function points
5. Non-functional SNAP points based on SNAP rules
6. COSMIC function points
7. FISMA function points
8. NESMA function points
9. Simple function points
10. Mark II function points
11. Unadjusted function points
12. Function points "light"
13. Engineering function points
14. Feature points
15. Use-case points
16. Story points
17. Lines of code (logical statements)
18. Lines of code (physical lines)
19. RICE objects
20. Micro function points
21. Logical code statements
22. Physical lines of code
23. Additional metrics as published

The pattern matching approach depends upon the availability of thousands of existing projects to be effective. However now that function point metrics have been in use for more than 38 years there are thousands of projects available.

One additional feature of pattern matching is that it can provide size data on requirements creep and on deferred functions. Thus the pattern-matching method predicts size at the end of the requirements phase, creeping requirements, size at delivery, and also the probable number of function points that might have to be deferred to achieve a desired delivery date.

In fact the pattern matching approach does not stop at delivery, but can continue to predict application growth year by year for up to 10 years after deployment.

The ability to size open-source and commercial applications or even classified weapons systems is a unique feature of sizing via pattern matching. Table 6 shows 100 applications sized via pattern matching with an average speed of about 1.8 minutes per application:

Table 6: Sizes of 100 Software Applications

	Applications *NOTE: SRM sizing takes about 1.8 minutes per application for sizing (patent-pending).*	Size in Function Points IFPUG 4.3	SNAP Non-function Points IFPUG	Size in Logical Code Statements
1	IBM Future System FS/1 (circa 1985 not completed)	515,323	108,218	68,022,636
2	Star Wars missile defense	352,330	42,280	32,212,992
3	World-wide military command and control (WWMCCS)	307,328	56,856	28,098,560
4	U.S. Air Traffic control	306,324	59,121	65,349,222
5	Israeli air defense system	300,655	63,137	24,052,367
6	North Korean Border defenses	273,961	50,957	25,047,859
7	Iran's air defense system	260,100	46,558	23,780,557
8	SAP	253,500	32,070	18,480,000
9	Aegis destroyer C&C	253,088	49,352	20,247,020
10	Oracle	229,434	29,826	18,354,720
11	Windows 10 (all features)	198,050	21,786	12,675,200
12	Obamacare web (all features)	107,350	5,720	12,345,250
13	Microsoft Office Professional 2010	93,498	10,285	5,983,891
14	Airline reservation system	38,392	5,759	6,142,689
15	North Korean Long-Range Missile controls	37,235	4,468	5,101,195
16	NSA code decryption	35,897	3,590	3,829,056
17	FBI Carnivore	31,111	2,800	3,318,515
18	FBI fingerprint analysis	25,075	3,260	2,674,637
19	NASA space shuttle	23,153	3,010	2,116,878
20	VA Patient monitoring	23,109	3,004	4,929,910
21	Data Warehouse			

		21,895	2,846	1,077,896
22	NASA Hubble controls	21,632	2,163	1,977,754
23	Skype	21,202	3,392	1,130,759
24	Shipboard gun controls	21,199	4,240	1,938,227
25	American Express billing	20,141	3,223	1,432,238
26	M1 Abrams battle tank operations	19,569	3,131	1,789,133
27	Apple I Phone v6 oprations	19,366	2,518	516,432
28	IRS income tax analysis	19,013	2,472	1,352,068
29	Cruise ship navigation	18,896	2,456	1,343,713
30	MRI medical imaging	18,785	2,442	1,335,837
31	Google search engine	18,640	2,423	1,192,958
32	Amazon web site	18,080	2,350	482,126
33	State wide child support	17,850	2,321	952,000
34	Linux	17,505	2,276	700,205
35	FEDEX shipping controls	17,378	2,259	926,802
36	Tomahawk cruise missile	17,311	2,250	1,582,694
37	Denver Airport luggage (original)	17,002	2,166	1,554,497
38	Inventory management	16,661	2,111	1,332,869
39	EBAY transaction controls	16,390	2,110	1,498,554
40	Patriot missile controls	16,239	2,001	1,484,683
41	IBM IMS data base	15,392	1,939	1,407,279
42	Toyota robotic manufacturing	14,912	1,822	3,181,283
43	Android operating system	14,019	1,749	690,152
44	Quicken 2015	13,811	1,599	679,939
45	State transportation ticketing	12,300	1,461	656,000
46	State Motor vehicle registrations	11,240	1,421	599,467
47	Insurance claims handling	11,033	1,354	252,191
48	SAS statistical package	10,927	1,349	999,065

49	Oracle CRM Features	10,491	836	745,995
50	DNA Analysis	10,380	808	511,017
51	EZPass vehicle controls	4,751	594	253,400
52	Cat scan medical device	4,575	585	244,000
53	Chinese submarine sonar	4,500	522	197,500
54	Microsoft Excel 2007	4,429	516	404,914
55	Citizens bank on-line	4,017	655	367,224
56	MapQuest	3,969	493	254,006
57	Bank ATM controls	3,917	571	208,927
58	NVIDIA graphics card	3,793	464	151,709
59	Lasik surgery (wave guide)	3,625	456	178,484
60	Sun D-Trace utility	3,505	430	373,832
61	Microsoft Outlook	3,450	416	157,714
62	Microsoft Word 2007	3,309	388	176,501
63	Adobe Illustrator	2,507	280	178,250
64	SpySweeper antispyware	2,227	274	109,647
65	Norton anti-virus software	2,151	369	152,942
66	Microsoft Project 2007	2,108	255	192,757
67	Microsoft Visual Basic	2,068	247	110,300
68	All-in-one printer	1,963	231	125,631
69	AutoCAD	1,900	230	121,631
70	Garmin hand-helc GPS	1,858	218	118,900
71	Intel Math function library	1,768	211	141,405
72	PBX switching system	1,658	207	132,670
73	Motorola cell phone contact list	1,579	196	144,403
74	Seismic analysis	1,564	194	83,393
75	Sidewinder missile controls	1,518	188	60,730
76	Apple I Pod	1,507	183	80,347

77	Property tax assessments	1,492	179	136,438
78	Mozilla Firefox (original)	1,450	174	132,564
79	Google Gmail	1,379	170	98,037
80	Digital camera controls	1,344	167	286,709
81	IRA account management	1,340	167	71,463
82	Consumer credit report	1,332	165	53,288
83	Sun Java compiler	1,310	163	119,772
84	All in one printer driver	1,306	163	52,232
85	Laser printer driver	1,285	162	82,243
86	JAVA compiler	1,281	162	91,096
87	Smart bomb targeting	1,267	150	67,595
88	Wikipedia	1,257	148	67,040
89	Casio atomic watch with compass, tides	1,250	129	66,667
90	Cochlear implant (embedded)	1,250	135	66,667
91	APAR analysis and routing	1,248	113	159,695
92	Computer BIOS	1,215	111	86,400
93	Automobile fuel injection	1,202	109	85,505
94	Anti-lock brake controls	1,185	107	63,186
95	Ccleaner utility	1,154	103	73,864
96	Hearing aid (multi program)	1,142	102	30,448
97	LogiTech cordless mouse	1,134	96	90,736
98	Instant messaging	1,093	89	77,705
99	Twitter (original circa 2009)	1,002	77	53,455
100	Denial of service virus	866	-	79,197
	Averages	**42,682**	**6,801**	**4,250,002**

Note: sizes assume IFPUG 4.3

Note: All sizes by Software Risk Master (SRM)

The ability to size open-source and commercial applications or even classified weapons systems is a unique feature of sizing via pattern matching and also unique to Software Risk Master (SRM).

No other sizing method can be used without access to at least published requirements. The unique patter-matching size technique of SRM is the only one that can size software without detailed inner knowledge. This is because SRM uses external patterns.

Note that SRM sizing is a proprietary trade secret and not available to the public. However a visit to the Namcook web site www.Namcook.com includes a trial version that is run-limited but can produce several project sizes before the limits are reached.

Early Risk Analysis

One of the main purposes of early sizing is to be able to identify software risks early enough to plan and deploy effective solutions. (This is why Namcook calls its sizing and estimating tool "Software Risk Master" (SRM).

If risks are not identified until after the requirements are complete, it is usually too late to make changes in development methods.

The 25 major risks where application size has been proven to be a major factor in application costs, schedules, and quality include but are not limited to:

Table 1: Software Risks Related to Application Size

1. Project cancellations
2. Project cost overruns
3. Project schedule delays
4. Creeping requirements (> 1% per month)
5. Deferred features due to deadlines (>20% of planned features)
6. High defect potentials
7. Low defect removal efficiency (DRE)
8. Latent security flaws in application when released
9. Error-prone modules (EPM) in applications
10. High odds of litigation for outsource contract projects
11. Low customer satisfaction levels
12. Low team morale due to overtime and over work
13. Inadequate defect tracking which fails to highlight real problems

14. Inadequate cost tracking which omits major expense elements
15. Long learning curves by maintenance and support teams
16. Frequent user errors when learning complex new systems
17. Post-release cyber-attacks (denial of service, hacking, data theft, etc.)
18. High cost of learning to use the application (COL)
19. High cost of quality (COQ)
20. High technical debt
21. High maintenance costs
22. High warranty costs
23. Excessive quantities of rework
24. Difficult enhancement projects
25. High total cost of ownership (TCO)

All 25 of these software risks are proportional to application size, so early sizing is a useful precursor for risk avoidance and risk mitigation. In estimating mode Software Risk Master (SRM) predicts the odds of these risks occurring, and in measurement mode can measure their impact on completed projects.

There are also many risks that are not directly related to project size: bankruptcies, theft of intellectual property, cyber attacks on applications, loss of key personnel, and many more. In total the Namcook Analytics LLC master list of current software risks includes a total of 210 software risk factors.

Lifetime Sizing with Software Risk Master™

Although this report concentrates on quality and the initial release of a software application, the Software Risk Master (SRM sizing algorithms actually create 15 size predictions. The initial prediction is for the nominal size at the end of requirements. SRM also predicts requirements creep and deferred functions for the initial release.

After the first release SRM predicts application growth for a 10 year period. To illustrate the full set of SRM size predictions, table 7 shows a sample application with a nominal starting size of 10,000 function points. All of the values are in round numbers to make the patterns of growth clear:

Table 7: SRM Multi-Year Sizing Example

Patent application 61434091. February 2012.

Nominal application size in IFPUG function points	**10,000**
SNAP points	**1,389**
Language	**C**
Language level	**2.50**
Logical code statements	**1,280,000**

		Function Points	SNAP Points	Logical Code	
1	Size at end of requirements	10,000	1,389	1,280,000	
2	Size of requirement creep	2,000	278	256,000	
3	Size of planned delivery	12,000	1,667	1,536,000	
4	Size of deferred features	-4,800	(667)	(614,400)	
5	Size of actual delivery	7,200	1,000	921,600	
6	Year 1 usage	12,000	1,667	1,536,000	Kicker
7	Year 1 usage	13,000	1,806	1,664,000	
8	Year 1 usage	14,000	1,945	1,792,000	
9	Year 1 usage	17,000	2,361	2,176,000	Kicker
10	Year 1 usage	18,000	2,500	2,304,000	
11	Year 1 usage	19,000	2,639	2,432,000	
12	Year 1 usage	20,000	2,778	2,560,000	
13	Year 1 usage	23,000	3,195	2,944,000	Kicker

14	Year 1 usage	24,000	3,334	3,072,000
15	Year 1 usage	25,000	3,473	3,200,000

Kicker = Extra features added to defeat competitors.

Note: Simplified example with whole numbers for clarity.

Note: Deferred features usually due to schedule deadlines.

As can be seen from table 7 software applications do not have a single fixed size, but continue to grow and change for as long as they are being used by customers or clients. Therefore productivity and quality data needs to be renormalized from time to time. Namcook suggests renormalization every at the beginning of every fiscal or calendar year.

Economic Modeling with Software Risk Master

Because Software Risk Master can predict the results of any methodology used for any size and kind of software project, it is in fact a general economic model that can show the total cost of ownership (TCO) and the cost of quality (COQ) for a variety of software development methods and practices.

For example SRM can show immediate results in less than one minute for any or all of the more than 60 developments; for any combination of 84 programming languages; and for work patterns in any of more than 50 countries.

The 20 most common methodologies used by SRM customers as of 2016 include in alphabetical order:

1. Agile development
2. Capability Maturity Model Integrated (CMMI)™ - all 5 levels
3. Extreme programming (XP)
4. Feature-driven development (FDD)
5. Formal inspections (combined with other methods)
6. Hybrid development (features from several methods)
7. Information Engineering (IE)
8. Iterative development
9. Lean software development (alone or in combination)

10. Mashup software development
11. Model-driven development
12. Open-source development models
13. Personal software process (PSP)
14. Rational unified process (RUP)
15. Reusable components and artifacts (various levels of reuse)
16. SCRUM (alone or with other methods)
17. Spiral development
18. Team software process (TSP)
19. Test driven development (TDD)
20. Waterfall development

It takes less than one minute to switch Software Risk Master from one methodology to another, so it is possible to examine and evaluate 10 to 20 alternatives methods in less than half an hour. (This is not a feature of most other parametric estimation tools.)

Software Risk Master can also model any level of development team experience, management experience, tester experience, and even client experience.

Software Risk Master can also show the results of any of 84 different programming language or combination of programming languages for more than 79 languages such as ABAP, Ada, APL, Basic, C, C#, C++, CHILL, COBOL, Eiffel, Forth, Fortran, HTML, Java, Javascript, Objective C, PERL, PHP, PL/I, Python, Ruby, Smalltalk, SQL, Visual Basic, and many other languages. In theory Software Risk Master could support all 2,500 programming languages, but there is very little empirical data available for many of these.

To add clarity to the outputs, Software Risk Master can show identical data for every case, such as showing a sample application of 1000 function points and then changing methods, programming languages, CMMI levels, and team experience levels. Using the same data and data formats allows side-by-side comparisons of different methods and practices.

This allows clients to judge the long-range economic advantages of various approaches for both development and total cost of ownership (TCO).

The Future of Sizing and Estimating Software with Function Points

Every year since 1975 more and more companies have adopted function point metrics; fewer and fewer companies are using lines of code, story points, cost per defect, and other ambiguous and hazardous metrics.

The governments of Brazil and Korea already use function points for government software contracts (Korea sent a delegation to Namcook Analytics to discuss this policy.) Other countries such as Italy and Malaysia are also planning to use function points for contracts (the author is an advisor to the Malaysian software testing organization and knows that Malaysia is considering function points for contracts).

Outside of the United States, the 25 countries with the most certified function point counters and the widest usage of function points among technology companies include:

Countries Expanding Use of Function Points 2016

1	Argentina	
2	Australia	
3	Belgium	
4	**Brazil**	**Required for government contracts**
5	Canada	
6	China	
7	Finland	
8	France	
9	Germany	
10	India	
11	**Italy**	**Required for government contracts**
12	**Japan**	**Required for government contracts**
13	**Malaysia**	**Required for government contracts**
14	Mexico	
15	Norway	
16	Peru	
17	Poland	
18	Singapore	
19	**South Korea**	**Required for government contracts**
20	Spain	
21	Switzerland	
22	Taiwan	
23	The Netherlands	
24	United Kingdom	
25	United States	

It is interesting that several countries with large numbers of technology companies have not utilized function point metrics to the same degree as the 25 countries shown above. Some of the countries that do not seem to have internal function point user groups as of 2016 (although this is uncertain) include in alphabetical order: China, Russia, Saudi Arabia, The Ukraine.

Because software is important in all countries and function points are the best metric for estimating and measuring software quality, costs, and productivity it can be expected by about 2025 that every industrial country in the world will use function point metrics and have internal function point user groups.

Even today in 2017 Namcook receives requests for function point data from over 45 countries per year including several such as China, Colombia, Cuba, Jordan, Pakistan, Russia, Saudi Arabia, and Viet Nam which are just starting to examine the usefulness of function point metrics.

For economic analysis and quality analysis of software, function points are the best available metric and already have more benchmark data than all other metrics combined.

Summary and Conclusions

Large software projects are among the most risky business ventures in history. The failure rate of large systems is higher than other kinds of manufactured products. Cost overruns and schedule delays for large software projects are endemic and occur on more than 75% of large applications. Indeed about 35% of large systems > 10,000 function points are cancelled and not delivered: one of the most expensive forms of business failure in history.

Early sizing via pattern matching and function point metrics combined with early risk analysis can improve the success rates of large software applications due to alerting mangers and software teams to potential hazards while there is still time enough to take corrective actions prior to expending significant funds.

References and Readings

Jones, Capers; A Guide to Selecting Software Measures and Metrics; CRC Press, 2017.

Jones, Capers; Software Methodologies, a Quantitative Guide, CRC Press, 2017.

Jones Capers; Quantifying Software: Global and Industry Perspectives; CRC Press, 2017.

Jones, Capers; The Technical and Social History of Software Engineering; Addison Wesley, 2014.

Jones, Capers and Bonsignour, Olivier; The Economics of Software Quality; Addison Wesley Longman, Boston, MA; ISBN 10: 0-13-258220—1; 2011; 585 pages.

Jones, Capers; Software Engineering Best Practices; McGraw Hill, New York, NY; ISBN 978-0-07-162161-8; 2010; 660 pages.

Jones, Capers; Applied Software Measurement; McGraw Hill, New York, NY; ISBN 978-0-07-150244-3; 2008; 662 pages.

Jones, Capers; Estimating Software Costs; McGraw Hill, New York, NY; 2007; ISBN-13: 978-0-07-148300-1.

Jones, Capers; Software Assessments, Benchmarks, and Best Practices; Addison Wesley Longman, Boston, MA; ISBN 0-201-48542-7; 2000; 657 pages.

Jones, Capers; Conflict and Litigation Between Software Clients and Developers; Software Productivity Research, Inc.; Burlington, MA; September 2007; 53 pages; (SPR technical report).

Additional Literature

The literature on function point metrics is quite extensive. Following are some of the more useful books:

Abran, Alain; Software Estimating Models; Wiley-IEEE Computer Society; 2015

Abran, Alain; Software Metrics and Metrology; Wiley-IEEE Computer Society; 2010

Abran, Alain; Software Maintenance Management: Evolution and Continuous Improvement; Wiley-IEEE Computer Society, 2008.

Abran, Alain and Dumke, Reiner R; Innovations in Software Measurement; Shaker-Verlag, Aachen, DE; ISBN 3-8322-4405-0; 2005; 456 pages.

Abran, Alain; Bundschuh, Manfred; Dumke, Reiner; Ebert; Christof; and Zuse, Horst; *Software Measurement News*; Vol. 13, No. 2, Oct. 2008 (periodical).

Bundschuh, Manfred and Dekkers, Carol; The IT Measurement Compendium; Springer-Verlag, Berlin, DE; ISBN 978-3-540-68187-8; 2008; 642 pages.

Chidamber, S.R. & Kemerer, C.F.; "*A Metrics Suite for Object-Oriented Design*"; IEEE Trans. On Software Engineering; Vol. SE20, No. 6; June 1994; pp. 476-493.

Dumke, Reiner; Braungarten, Rene; Büren, Günter; Abran, Alain; Cuadrado-Gallego, Juan J; (editors); Software Process and Product Measurement; Springer-Verlag, Berlin; ISBN 10: 3-540-89402-0; 2008; 361 pages.

Ebert, Christof and Dumke, Reiner; Software Measurement: Establish, Extract, Evaluate, Execute; Springer-Verlag, Berlin, DE; ISBN 978-3-540-71648-8; 2007; 561 pages.

Gack, Gary; Managing the Black Hole: The Executives Guide to Software Project Risk; Business Expert Publishing, Thomson, GA; 2010; ISBN10: 1-935602-01-9.

Gack, Gary; *Applying Six Sigma to Software Implementation Projects*; http://software.isixsigma.com/library/content/c040915b.asp.

Galorath, Dan and Evans, Michael; Software Sizing, Estimation, and Risk Management; Auerbach Publications, Boca Raton, FL; 2006.

Garmus, David & Herron, David; Measuring the Software Process: A Practical Guide to Functional Measurement; Prentice Hall, Englewood Cliffs, NJ; 1995.

Garmus, David and Herron, David; Function Point Analysis – Measurement Practices for Successful Software Projects; Addison Wesley Longman, Boston, MA; 2001; ISBN 0-201-69944-3;363 pages.

Gilb, Tom and Graham, Dorothy; Software Inspections; Addison Wesley, Reading, MA; 1993; ISBN 10: 0201631814.

Harris, Michael D.S., Herron, David, and Iwanicki, Stasia; The Business Value of IT; CRC Press, Auerbach; Boca Raton, FL; 2008; ISBN 978-14200-6474-2.

International Function Point Users Group (IFPUG); IT Measurement – Practical Advice from the Experts; Addison Wesley Longman, Boston, MA; 2002; ISBN 0-201-74158-X; 759 pages.

Kemerer, C.F.; "*Reliability of Function Point Measurement - A Field Experiment*"; Communications of the ACM; Vol. 36; pp 85-97; 1993.

Parthasarathy, M.A.; Practical Software Estimation – Function Point Metrics for Insourced and Outsourced Projects; Infosys Press, Addison Wesley, Upper Saddle River, NJ; 2007; ISBN 0-321-43910-4.

Putnam, Lawrence H.; Measures for Excellence -- Reliable Software On Time, Within Budget; Yourdon Press - Prentice Hall, Englewood Cliffs, NJ; ISBN 0-13-567694-0; 1992; 336 pages.

Putnam, Lawrence H and Myers, Ware.; Industrial Strength Software - Effective Management Using Measurement; IEEE Press, Los Alamitos, CA; ISBN 0-8186-7532-2; 1997; 320 pages.

Royce, Walker; Software Project Management – Unified Framework; Addison Wesley, Boston, MA; 1999.

Stein, Timothy R; The Computer System Risk Management Book and Validation Life Cycle; Paton Press, Chico, CA; 2006; ISBN 10: 1-9328-09-5; 576 pages.

Stutzke, Richard D; Estimating Software-Intensive Systems; Addison Wesley, Upper Saddle River, NJ; 2005; ISBN 0-201-70312-2; 918 pages.