The Costs, Accuracy, and Uses of Software Benchmarks

Capers Jones, CTO
Namcook Analytics LLC

January, 2018

Abstract

What is called "historical benchmark data" for software projects has proven to be incomplete and incorrect unless the data is validated by interviews with project teams. The most common omissions are unpaid overtime, management effort, all forms of client effort, and the work of part-time specialists such as architects, business analysts, technical writers, function point counters, and integration specialists.

Quality data is even worse and routinely omits all defects found prior to testing, and sometimes even test defects are not recorded. The following table shows that an average "software benchmark" only contains about 37% of the full and true effort expended on a typical software project:

Measured Effort Versus Actual Effort
Development Projects

		Percent of Total	Measured Results
1	Business analysis	1.25%	
2	Risk analysis/sizing	0.26%	
3	Risk solution planning	0.25%	
4	Requirements	4.25%	
5	Requirement. Inspection	1.50%	
6	Prototyping	2.00%	
7	Architecture	0.50%	
8	Architecture. Inspection	0.25%	
9	Project plans/estimates	0.25%	
10	Initial Design	5.00%	
11	Detail Design	7.50%	7.50%
12	Design inspections	2.50%	
13	Coding	22.50%	22.50%
14	Code inspections	20.00%	
15	Reuse acquisition	0.03%	

16	Static analysis	0.25%	
17	COTS Package purchase	0.03%	
18	Open-source acquisition.	0.03%	
19	Code security audit.	0.25%	
20	Ind. Verif. & Valid.	1.00%	
21	Configuration control.	1.00%	
22	Integration	0.75%	
23	User documentation	2.00%	
24	Unit testing	0.75%	0.75%
25	Function testing	1.25%	1.25%
26	Regression testing	1.50%	1.50%
27	Integration testing	1.00%	1.00%
28	Performance testing	0.50%	
29	Security testing	0.50%	
30	Usability testing	0.75%	
31	System testing	2.50%	2.50%
32	Cloud testing	0.50%	
33	Field (Beta) testing	0.75%	
34	Acceptance testing	1.00%	
35	Independent testing	1.50%	
36	Quality assurance	2.00%	
37	Installation/training	0.65%	
38	Project measurement	0.50%	
39	Project office	1.00%	
40	Project management	10.00%	
	Cumulative Results	**100.00%**	**37.00%**

These gaps and omissions from software benchmarks are professionally embarrassing. Software is one of the most expensive and labor-intensive technical products in human history. Software economic analysis should be 100% complete, but in real life it is usually below 50% complete.

This paper discusses the costs, accuracy, and uses of several methods of collecting historical data. A new method is introduced which uses the Software Risk Master (SRM) parametric estimation tool during the measurement process to perform side-by-comparisons of historical data and predicted results. If there are significant differences between the SRM predicted values and historical results, then deeper analysis can be carried out or the SRM data can be used to fill in gaps and missing data. This approach is named "synthetic benchmarks"

This new method is similar to techniques used during medical examinations and laboratory tests for blood samples. In other words collections of standard values are compared against a patient's blood sample to find out if blood sugar or cholesterol are within average ranges or higher or lower than average. For software the parametric estimation tool provides the set of standard values for activity-based costs.

This new method of benchmark data collection has the potential of greatly improving benchmark accuracy while simultaneously improving the speed of benchmark data collection. It also opens a path for remote benchmark data collection with much higher accuracy than ordinary paper questionnaires.

Web: www.Namcook.com
Email: Capers.Jones3@Gmail.com

INTRODUCTION

As the developer of 12 commercial and proprietary software cost estimating tools, the author is often asked what seems to be a straight-forward question: "*How accurate are the estimates compared to historical data?*"

This is a reasonable question and a legitimate one. Every estimating tool vendor should be prepared to answer it. But the answer to this question is surprising. Usually the estimates from a modern commercial parametric estimation tool such as Software Risk Master (SRM) and the others are more accurate than most clients' historical data.

What the clients believe to be historical data is incomplete and omits much of the actual costs and work effort that were accrued. There are some exceptions to this rule:

- Data collected for defense and military projects where the contracts require precise cost collection.
- Data collected from contracts billed on a time and materials basis.
- Data collected from projects developed under a profit-center model.
- Data collected by a few leading companies such as IBM.

However cost and resource data for internal projects developed under a cost-center model range from having zero data collected to having only a percentage of total data collected. The author has met a number of project managers who are proud of the fact that their development teams do not measure. The managers assert that lack of measures aids morale.

Some software development methodologies such as Agile also lack effective measures. Some Agile projects don't measure at all, while other use quirky and non-standard metrics such as "story points" and "burn down" which lack consistency from team to team and company to company. In fact Agile projects are among the least accurate of any known methodology. Agile metrics such as story points have no ISO standards and vary by hundreds of percent. Agile sprints are hard to measure and not directly comparable to other methods. The author's technique for Agile using or SRM tool is to convert story points into function points and to consolidate sprints into a standard chart of accounts. This allows side-by-side comparisons between Agile and other methodologies.

Quality data is incomplete too. Most companies do not even start measuring quality until after unit test so all requirement and design defects are excluded, as are unit test defects. The result is a defect count that understates the true numbers of bugs by more than 60%. In fact some companies don't measure defects until after release of the software.

Thus when the outputs from accurate commercial software cost estimating tools such as Software Risk Master (SRM) are compared to what is called "historical data" the results tend to be alarming to clients.

The outputs from the estimating tools often indicate higher costs, more effort, longer schedules, and lower quality than the historical data indicates. It is seldom realized that the difference is because of major gaps and omissions in the historical data itself, rather than because of errors in the estimates.

It is fair to ask if historical data is incomplete, how is it possible to know the true amounts and judge the quantity of missing data that was omitted?

In order to correct the gaps and omissions that are normal in cost tracking systems, it is necessary to interview the development team members and the project managers. The interviews can either be face to face or remote.

During these interview sessions, the contents of the historical data collected for the project are compared to a complete work breakdown structure derived from similar projects. For each activity and task that occurs in the work breakdown structure, but is missing from the historical data, the developers are asked whether or not the activity occurred. If it did occur, the developers are asked to reconstruct from memory or their informal records the number of hours that the missing activity accrued.

Problems with errors and "leakage" from software cost tracking systems are as old as the software industry itself. The first edition of the author's book Applied Software Measurement was published in 1991. The third edition was published in 2008. Yet the magnitude of errors in cost tracking systems is essentially the same today in 2018 as it was in 1991. Following is an excerpt from the first edition:

"*It is a regrettable fact that most corporate tracking systems for effort and costs (dollars, work hours, person months, etc.) are incorrect and manage to omit from 30% to more than 70% of the real effort applied to software projects. Thus most companies cannot safely use their own historical data for predictive purposes.*"

These gaps and omissions in tracking historical effort, progress, and costs are surprising since there are more than 50 project tracking tools on the market, and about 25 such tools from open-source providers. Yet in spite of the plethora of available tools, many companies don't use any of these for their software projects.

Gaps and Omissions for Software Development Data

For contract software where charges are on a time and material basis, vendors are motivated to keep good records because otherwise they won't get paid.

For many companies that develop their own software internally, software development operates as a cost center. That is, costs are not charged to user departments or operating units but come from special corporate or divisional accounts. In this situation poor measurement practices are endemic and hardly anyone knows how much effort goes into software projects.

If software development operates as a profit center and charges operating units for applications, then there is some motivation for keeping at least fairly good records. However there are still gaps such as unpaid overtime. Also, some profit centers only charge for technical development and not for management.

If a software project is for a military organization and uses earned-value analysis, then cost data will probably be fairly complete and reliable. Even here unpaid overtime is often omitted.

The commonest omissions from historical resource and cost data are shown in table 1 for ordinary civilian applications as developed in banks, insurance companies, manufacturing and other normal businesses that build software under a cost-center model.

Table 1 uses a standard chart of accounts with 40 activities developed by the author for activity-based cost analysis (ABC). This table is used by the author to check the completeness of benchmarks during client interviews. Table 1 assumes an ordinary application of about 1000 function points in size:

Table 1: Measured Effort Versus Actual Effort
Development Projects

		Percent of Total	Measured Results
1	Business analysis	1.25%	
2	Risk analysis/sizing	0.26%	
3	Risk solution planning	0.25%	
4	Requirements	4.25%	
5	Requirement. Inspection	1.50%	
6	Prototyping	2.00%	
7	Architecture	0.50%	
8	Architecture. Inspection	0.25%	
9	Project plans/estimates	0.25%	
10	Initial Design	5.00%	
11	Detail Design	7.50%	7.50%
12	Design inspections	2.50%	
13	Coding	22.50%	22.50%

14	Code inspections	20.00%	
15	Reuse acquisition	0.03%	
16	Static analysis	0.25%	
17	COTS Package purchase	0.03%	
18	Open-source acquisition.	0.03%	
19	Code security audit.	0.25%	
20	Ind. Verif. & Valid.	1.00%	
21	Configuration control.	1.00%	
22	Integration	0.75%	
23	User documentation	2.00%	
24	Unit testing	0.75%	0.75%
25	Function testing	1.25%	1.25%
26	Regression testing	1.50%	1.50%
27	Integration testing	1.00%	1.00%
28	Performance testing	0.50%	
29	Security testing	0.50%	
30	Usability testing	0.75%	
31	System testing	2.50%	2.50%
32	Cloud testing	0.50%	
33	Field (Beta) testing	0.75%	
34	Acceptance testing	1.00%	
35	Independent testing	1.50%	
36	Quality assurance	2.00%	
37	Installation/training	0.65%	
38	Project measurement	0.50%	
39	Project office	1.00%	
40	Project management	10.00%	
	Cumulative Results	**100.00%**	**37.00%**
	Unpaid overtime	7.50%	

As can be seen, only 7 out of 40 activities are included in normal internal "historical data" collections. The total quantity of work measured is only about 37% of the real effort expended. Low-level design, coding, and testing are the primary activities that get recorded. These activities usually amount to less than 50% of total effort and in extreme cases to less than 35% of total effort.

Even worse would be the common measurement set of "design, code, and unit test" commonly abbreviated to DCUT. It is unprofessional to measure only part of the effort for software development.

The mathematical result of gaps and omissions in historical data is to balloon apparent productivity rates. In table 1 the apparent productivity rate might be 15 function points per staff month but real productivity would only be about 6 function points per staff month if the missing activities were included.

Unpaid overtime is a special case. Unpaid overtime shortens schedules and lowers costs, but is essentially invisible to most companies because it is never recorded. For

commercial software vendors unpaid overtime averages about 4 hours per week, or 10% of the nominal work week. Omitting unpaid overtime introduces a distortion into software productivity data of about 10%. In other words apparent productivity might seem to be 10 function points per staff month but is really less than 9 function points per staff month if the missing hours for unpaid overtime were included.

Gaps and Omissions for Software User Effort and Cost Data

The most common omissions from software economic studies are the costs and defects contributed by the users themselves. Of course for commercial and embedded software users are not active participants, other than focus groups or field trials.

However for internal information technology projects, users contribute a majority of effort for requirements, and significant amounts of effort later in activities such as change control and acceptance testing. Indeed for Agile projects at least one future user is assigned full-time to the development teams and therefore comprises perhaps 20% of the development effort.

For ordinary information systems applications developed internally for a company's own use, user effort is a large but invisible cost element. Table 2 shows the major activities where users participate and the approximate effort utilized:

Table 2: User Effort Versus Development Team Effort Development Projects

		Team Percent of Total	User Percent of Total
1	Business analysis	1.25%	3.75%
2	Risk analysis/sizing	0.26%	
3	Risk solution planning	0.25%	
4	Requirements	4.25%	5.31%
5	Requirement. Inspection	1.50%	1.50%
6	Prototyping	2.00%	0.60%
7	Architecture	0.50%	
8	Architecture. Inspection	0.25%	
9	Project plans/estimates	0.25%	
10	Initial Design	5.00%	
11	Detail Design	7.50%	
12	Design inspections	2.50%	
13	Coding	22.50%	
14	Code inspections	20.00%	
15	Reuse acquisition	0.03%	
16	Static analysis	0.25%	
17	COTS Package purchase	0.03%	1.00%
18	Open-source acquisition.	0.03%	
19	Code security audit.	0.25%	

20	Ind. Verif. & Valid.	1.00%	
21	Configuration control.	1.00%	
22	Integration	0.75%	
23	User documentation	2.00%	1.00%
24	Unit testing	0.75%	
25	Function testing	1.25%	
26	Regression testing	1.50%	
27	Integration testing	1.00%	
28	Performance testing	0.50%	
29	Security testing	0.50%	
30	Usability testing	0.75%	
31	System testing	2.50%	
32	Cloud testing	0.50%	
33	Field (Beta) testing	0.75%	9.00%
34	Acceptance testing	1.00%	4.00%
35	Independent testing	1.50%	
36	Quality assurance	2.00%	
37	Installation/training	0.65%	9.75%
38	Project measurement	0.50%	
39	Project office	1.00%	
40	Project management	10.00%	
	Cumulative Results	**100.00%**	**35.91%**
	Unpaid overtime	5.00%	5.00%

As can be seen, the total quantity of user effort on typical internal applications of about 1000 function points in size is a bit more than one third of the total effort. However user costs are seldom included in software cost estimates or budgets, and are seldom included in corporate cost tracking systems either.

You can expect user effort to exceed 5 hours per function point on normal waterfall projects of 1000 function points in size. For Agile projects of 1000 function points in size user effort might top 7 hours per function point.

Gaps and Omissions in Software Quality Data

When we look at software quality data, we see similar leakages. Many companies do not track any bugs before release. Only sophisticated companies such as IBM, Raytheon, Motorola, and the like track pre-test bugs. At IBM there were even volunteers who recorded bugs found during desk check sessions, debugging, and unit testing, just to provide enough data for statistical analysis. Table 3 shows the pattern of missing data for software defect measurements:

Table 3: Measured Quality Versus Actual Quality Development Projects

		Defects Removed	Defects Measured
1	Requirements inspection	75	
2	Requirements changes	10	
3	Architecture inspection	15	
4	Initial design inspection	50	
5	Detail design inspection	100	
6	Design changes	20	
7	Code inspections	350	
8	Code changes	75	
9	User document editing	25	
10	User document changes	10	
11	Static analysis	235	
12	Unit test	50	
13	Function testing	50	50
14	Regression testing	25	25
15	Integration testing	75	75
16	Performance testing	25	25
17	Security testing	10	10
18	Usability testing	20	20
19	System testing	50	50
20	Cloud testing	10	10
21	Field (Beta) testing	20	20
22	Acceptance testing	15	15
23	Independent testing	10	10
24	Quality assurance	25	25
25	Customer bug reports	150	150
	Cumulative Results	**1,500**	**485**
	Percent of Total Defects	**100.00%**	**32.33%**

Out of 25 total forms of defect removal, data is only collected for 13 of these under normal conditions. The apparent defect density of the measured defects is less than one third of the true volume of software defects. In other words true defect potentials would be about 5 defects per function point, but due to gaps in the measurements apparent defect potentials would seem to be only about 2 defects per function point.

For the software industry as a whole the costs of finding and fixing bugs is the #1 cost driver. It is professionally embarrassing for the industry to be so lax about measuring the most expensive kind of work since software began.

Gaps and Omissions in Software Maintenance Data

The word "maintenance" is highly ambiguous and can include 25 different kinds of work. This makes maintenance benchmarks very difficult to perform at all, and even more difficult to perform with accuracy. The various kinds of work subsumed under the word "maintenance" include:

1. Major Enhancements (new features of > 20 function points)
2. Minor Enhancements (new features of < 5 function points)
3. Maintenance (repairing defects for good will)
4. Warranty repairs (repairing defects under formal contract)
5. Customer support (responding to client phone calls or problem reports)
6. Error-prone module removal (eliminating very troublesome code segments)
7. Mandatory changes (required or statutory changes)
8. Complexity or structural analysis (charting control flow plus complexity metrics)
9. Code restructuring (reducing cyclomatic and essential complexity)
10. Optimization (increasing performance or throughput)
11. Migration (moving software from one platform to another)
12. Conversion (Changing the interface or file structure)
13. Reverse engineering (extracting latent design information from code)
14. Reengineering (transforming legacy application to modern forms)
15. Dead code removal (removing segments no longer utilized)
16. Dormant application elimination (archiving unused software)
17. Nationalization (modifying software for international use)
18. Mass updates such as Euro or Year 2000 Repairs
19. Refactoring, or reprogramming applications to improve clarity
20. Retirement (withdrawing an application from active service)
21. Field service (sending maintenance members to client locations)
22. Reporting bugs or defects to software vendors
23. Installing updates received from software vendors
24. Processing invalid defect reports
25. Processing duplicate defect reports

Table 4 illustrates the approximate annual percentage of maintenance costs for these 25 separate kinds of work and then the percentage that are likely to be included in maintenance benchmarks:

Table 4: Gaps and Omissions in Software Maintenance Benchmarks

		Percent of Maintenance	Included in Benchmarks
1	Major enhancements	11.00%	12.00%
2	Minor enhancements	11.00%	
3	Defect repairs	13.00%	13.00%
4	Warranty repairs	3.00%	3.00%
5	Customer support	15.00%	15.00%
6	Error-prone module removal	3.00%	
7	Mandatory changes	4.00%	
8	Complexity analysis	0.50%	
9	Code restructuring	0.50%	
10	Optimization	3.00%	
11	Migration	4.00%	
12	File conversion	3.00%	
13	Reverse engineering	2.00%	
14	Reengineering	3.00%	
15	Dead code removal	1.00%	
16	Dormant application removal	1.00%	
17	Nationalization	2.00%	
18	Mass updates	3.00%	
19	Refactoring	3.00%	
20	Retirement of application	1.00%	
21	Field service	1.00%	1.00%
22	Bug reports to related packages	1.00%	
23	Vendor update installation	2.00%	
24	Invalid defect processing	4.00%	
25	Duplicate defect processing	5.00%	
	TOTAL	100.00%	44.00%

As can be seen normal benchmarks only record about 44% of the actual effort that is defined under the very ambiguous word “maintenance.”

It is interesting that while major enhancements are usually included in benchmarks, minor enhancements are invisible and almost never measured even though in sum they are quite expensive. There is an interesting reason for this.

The normal rules for function point analysis have lower limits due to the mathematical methods used for adjusting complexity. Function points by the International Function Point Users Group (IFPUG) don’t count well for small changes below about 10 function points in size. Most small enhancements are between 1 and 5 function points in size.

Because they are so small it is difficult to normalize the results. Also, the actual effort for small enhancements may be only a day or even a few hours. Individually small enhancements are comparatively unimportant, but if there are hundreds or thousands of

them in the course of a year, the cumulative costs are rather significant.

(Note: the author has a patented high-speed sizing method that size application whose size is less than 1 function point to as large as 350,000 function points. Sizing takes about 90 seconds per application regardless of its actual size.)

Why do Companies use Benchmarks and Measure Software Projects?

It is interesting to consider the business reasons why companies want to collect historical data from completed software projects. Having collected data on thousands of projects from hundreds of companies, here are the 10 main reasons given to the author by benchmark measurement clients:

1. **To avoid having software development outsourced**: Among the most common reasons for commissioning a measurement study is fear of losing jobs to outsourcing. Since top executives at the CEO and CFO levels have a low regard for many software development groups, there is often apprehension that software operations will be turned over to an outsource vendor. Software managers commission measurements to demonstrate to their higher management that they are capable. In these studies the client is usually a CIO or divisional software executive. These studies are typically large scale and may include new development, enhancements, migration, and annual maintenance.

2. **To establish starting points for process improvements**: Many companies are embarking on software process improvements or switching from waterfall to Agile. Often they will commission benchmark studies to establish their initial quantitative starting points. These studies usually include 10 projects divided between new development and enhancements. Later in about a year they will commission a follow on study to show the impact and value of the improvements.

3. **To compare effectiveness against competitors**: Top executives such as CIO's and CTO's are interested in how well competitors can build and maintain software. Therefore measurements are commissioned in order to rank internal software against industry norms. These projects tend to concentrate on new development since that is where competition is most intense. They usually include 10 projects for a single-site company and as many as 50 for a multi-site multi-national corporation.

4. **To validate commercial estimating tools:** Prior to acquiring a commercial estimation tool many companies will want to compare their historical data against predicted outcomes. Once the estimation vendors point out how inaccurate historical data usually is, companies may commission studies of several applications. These studies are usually small and include only a few projects.

5. **To validate internal estimating methods:** Since internal estimation methods are very common, it sometimes happens that companies commission measurement studies to validate their own proprietary estimation methods. These studies are usually small and include only a few projects.

6. **To help is selecting new methodologies:** Methodology selection resembles joining a cult. Companies select the most popular current methods such as Agile and don't bother to find out whether or not they are effective. Some companies are more sophisticated, and actually study various methods and even experiment with them. With these more sophisticated groups, measurements are used to validate the effectiveness of software methodologies. Since there are more than 30 methodologies, these studies will usually focus on the top candidates such as Agile, RUP, and TSP.

7. **To help in selecting new tools:** There are hundreds of software tool vendors and thousands of software tools, and all of the tool vendors make wonderful claims. The more sophisticated tool companies will commission controlled studies and collect data to show the impacts of their tools on quality and productivity. The less scrupulous tool companies simply assert something like "*Buy our Panacea/1 tool and get 10 to 1 improvements*." It is an unfortunate practice that these vendors do not provide any quantitative data and do not identify where the claims were validated. In fact most of the claims are spurious. Sophisticated companies will commission due diligence studies for project management tools, estimating tools, static analysis tools, and project tracking tools.

8. **To demonstrate professionalism:** Compared to law and medicine both of which keep very good records, software is notoriously lax in producing accurate estimates and in keeping accurate records of costs, effort, accomplishments and earned value. Yet for those in the software field who strive to achieve a more professional work style, keeping track of effort and results is a normal professional task with significant value. Accurate history leads to accurate predictions. Solid data raises the stature of the software community in the eyes of senior executives.

9. **To compete internally with rival business units**: Corporate politics can be severe and many large companies resemble the Court of the Borgias in the way executives interact. Rivalries between business units and between individual executives are quite common. If one group has made progress with software methods, practices, or tools they sometimes want an independent measurement group to provide quantitative data. This data might be used against rivals or to prevent rivals from using their own quantitative data to gain internal competitive advantages.

10. **To demonstrate the value of the Capability Maturity Model (CMMI®):** Although the CMMI is effective in identifying key process indicators (KPI) it has not been very effective in quantitative measures. If substantial funds are being

invested in ascending to higher levels, it is natural for the CEO, CFO, CTO and other executives to want quantitative proof that the higher levels of the CMMI yield tangible benefits. (Because the Software Engineering Institute (SEI) did not have quantitative data on this topic, the author received a contract from the Air Force to measure the value of the higher levels.)

There are also other reasons in addition to these 10, but 10 is enough to show that measurement of historical data is interesting to both senior corporate executives and also to software managers and software management executives.

Parametric Estimation and Synthetic Benchmark Data Collection

If the outputs of a software parametric estimating tool are more accurate than what is called historical data, why not use a parametric estimating tool to help improve benchmark measurement accuracy? This is the essential idea of parametric benchmarks. We call these "synthetic benchmarks" because predicted data from our SRM tool is used to fill gaps where clients have not collected actual data. The most common gaps and omissions include management effort, business analysis, technical writers, quality assurance, and project office staff effort.

This new way of collecting historical data involves high-speed and low-cost function point analysis using SRM combined with using SRM parametric estimates to help validate historical data. This new method is called "synthetic benchmarks." This new method collects quantitative benchmark data as described in this paper. The same method can be used for collecting qualitative assessment data, but that is a separate topic and not discussed here.

As mentioned previously normal function point analysis proceeds at an average rate of about 400 function points per day. The author has developed a new and proprietary high-speed function point method that can size more than 25 applications per day when used by experienced personnel. Assuming each of these 25 applications averages 400 function points in size, the daily rate would be 10,000 function points per day. However SRM has also sized large applications such as Windows 10 in less than 5 minutes, and this application tops 100,000 function points in size. This new method is embedded in our tool called Software Risk Master™ (SRM). With SRM the speed of function point sizing drops to about 90 seconds and the cost drops to pennies per function point.

Since remote self-reported benchmarks have gaps and omissions and on-site benchmarks are expensive, there is a new form of benchmark data collection that combines very high accuracy with very low cost.

Using this new method the benchmark questionnaire would be based on the input questionnaire to a parametric estimation tool such as Software Risk Master (SRM). The questionnaire would be available to benchmark clients.

Once the initial input questions have been answered the tool would generate both the function point size of the application and also provide a detailed estimate of the schedules, effort, staffing, and costs of the project using an activity-based cost structure. These estimates include the elements that are often omitted from benchmarks such as unpaid overtime and management costs.

With the Software Risk Master (SRM) tool as an example, sizing an application and generating a full estimate normally takes between 4 and 7 minutes.

Having a full estimate available as a basis for benchmark data collection will prompt users to include all relevant information and not ignore topics such as unpaid overtime, or project management.

When used in measurement mode, the estimation tool would allow users to perform any of these four actions for every activity displayed:

1. *Accept* the predicted value if it matches historical data

2. *Modify* the predicted value so it matches historical data exactly

3. Inform the estimation/measurement tool that certain predicted activities were not in fact performed (i.e. inspections, static analysis, etc.) by specifying *not used*.

4. Inform the estimation/measurement tool that a new kind of activity not included in the standard chart of account was performed by specifying *add activity*.

This form of benchmark would only take about 7 minutes to complete the questionnaire. Assuming that 3 people discussed the project before attempting to answer the questions, the net effort would probably be 3 people at 15 minutes each, or a total of 45 minutes per project. At internal costs of $50 per hour this would cost about $37.50

By merely scrolling through the predicted values from the estimate and accepting or modifying them, very accurate historical could be gathered. Even better this form of measurement could help to calibrate the estimates themselves. When used in predictive mode Software Risk Master™ displays a set of software benchmark parameters; i.e. staff, effort, schedule, and costs for each activity:

Table 5: Software Risk Master™ in Benchmark Prediction Mode

Activities	Staff	Effort Months	Schedule Months	Project Costs
Requirements	2.05	3.91	1.91	$39,127
Design	2.57	6.26	2.43	$62,604
Coding	5.39		6.51	$350,812

		35.08		
Testing	4.41	19.65	4.46	$196,455
Documentation	1.28	2.85	2.23	$28,456
Quality Assurance	1.12	3.56	3.17	$35,570
Management	1.21	12.52	13.73	$125,208
Totals	**6.11**	**83.82**	**20.71**	**$838,232**
Gross schedule months			20.71	
Overlap %			66.29%	
Predicted delivery schedule and date			**13.73**	

When used in measurement mode the Software Risk Master™ tool has a "micrometer" function that allows very high precision measurements to better than 1% precision. The micrometer mode uses three special parameters, assignment scopes, production rates, and overlap.

The assignment scope is the amount of work assigned to one person. The production rate is the amount of work performed in a fixed time interval such as one month. The overlap is the amount of an activity still not finished when the next activity begins. These three parameters are used in measurement mode to achieve very high precision:

Table 6: Software Risk Master™ in Parametric Measurement Mode

Activities	**Assignment Scope**	**Production Rate**	**Staff**	**Effort Months**	**Schedule Months**
Requirements	483	192	2.05	3.91	1.91
Design	362	120	2.57	6.26	2.43
Coding	8,186	1,140	5.39	35.08	6.51
Testing	10,232	2,036	4.41	19.65	4.46
Documentation	965	264	1.28	2.85	2.23
Quality Assurance	1,207	211	1.12	3.56	3.17
Management	1,062	60	1.21	12.52	13.73
Totals	**123**	**11.93**	**6.11**	**83.82**	**20.71**

Gross schedule months	20.71
Overlap %	66.29%
Predicted delivery months	**13.73**

By adjusting the assignment scope and production rate parameters (and the third parameter for schedule overlaps) historical project data can be matched exactly.

It would still be desirable to have some external validation of the benchmark data before it is accepted. This might consist of a short telephone call with a benchmark consultant or possibly an exchange of emails or an on-line chat.

This new method of parametric benchmarks simultaneously lowers the high costs of function point analysis and speeds up the collection of data, while providing standard data elements to users in order to help avoid common omissions.

The new parametric benchmark method combines several attractive features, including ease of use, low cost, and rapid results. Parametric benchmarks might easily top 100 projects per month or 1,200 per year in the United States, and perhaps 300 per month or 3,600 projects per year globally.

Measurement accuracy cannot achieve 100% precision or capture every single cost element. Too many unpredictable events such as layoffs of stakeholders, weather delays, illness of key personnel, and other transient phenomena can degrade measurement precision.

Measuring Brand New Tools, Methods, and Programming Languages

New software development methodologies occur about every 8 months. The author's current collection of software methodologies circa 2017 now has 70 named methodologies. These include Agile, container development DevOps, Extreme Programming (XP), mashups, Object-Oriented programming, the Rational Unified Process (RUP), the Team Software Process (TSP), iterative development, waterfall development, V-model development, Prince2, Merise, and many others. In addition there are hundreds of hybrid variations that use portions of several methods.

New programming languages have been coming out a rate of about one per month since the 1970's. As of 2018 there are more than 3,000 programming languages.

New tools have been coming at a rate of about one per week for the past 15 years. As of 2017 there are 25 major categories of software management and technical tools, and a total tool inventory that probably tops 5,000 commercial tools.

Capturing data on brand new methods and tools is a challenge for software benchmark providers. For one thing brand new tools or languages have no historical data of any kind and sometimes no users other than the original developers.

The solution developed by the author is to provide general "boxes" for various kinds of tools and methods. For example a partial list of project management tool categories in alphabetical order includes:

1. Cost estimation tools
2. Cost tracking tools
3. Defect tracking tools
4. Human resource assessment tools
5. Project management tools
6. Quality estimation tools
7. Project tracking tools

A partial list of software development tools in alphabetical order includes:

1. Assemblers
2. Compilers
3. Configuration control
4. Debuggers
5. Inspection support
6. Interpreters
7. Integration
8. Static analysis

Users will select the appropriate category and then enter the actual name of the tool or languages used.

For example if you are using a brand new programming language called "JavaDecafe" you would select compilers as the tool category.

When brand new tools or languages are being measured for the very first time, it is not possible to predict their impact with high precision. It is usually necessary to collect empirical data on perhaps a dozen uses of the new tool or language to form a preliminary judgment. Of course if a new programming language is just another variant of C or Java you can make reasonable assumptions as to its impact. But if it is truly new and has unique features, they need to be measured.

Analysis of the results of new tools, languages, and methods is of necessity a fairly complicated activity that requires careful collection of data.

As an example a new tool from a group called IntegraNova is entering the U.S. market. The company is headquartered in Spain and current clients are in Europe. The tool is a requirements modeling engine that feeds into an application generator.

It would be useful for IntegraNova to know the productivity and quality results from using this tool in a U.S. context. From European data it is possible to compare results against other methods such as Agile and RUP. However until there are U.S. projects and careful measurements, the results are speculative rather than definitive.

Life Expectancy of Software Benchmark Data

Once benchmark data is collected and entered into a data base, how long will it be useful? In other words, is data collected in 1995 still relevant in 2018? As it happens the value of historical benchmark data is just about as good as the value of historical medical records. Historical data allows longitudinal studies that can show improvements over long time spans.

It is interesting that waterfall projects of 1000 function points developed in 1985, 1990, 1995, 2000, 2005 and 2010 are remarkably similar in their productivity and quality rates and resemble waterfall projects delivered in 2017. On the other hand, projects using Agile, Rational Unified Process (RUP), and Team Software Process (TSP) all show higher productivity rates and better quality levels than waterfall at any year.

Thus historical data can reveal interesting trends about the effectiveness of new methods by allowing them to be compared against historical records for similar classes and types of applications.

When comparing modern projects developed in 2017 it is possible to limit the years for comparative projects and have a cut-off point of perhaps 2010. However, calendar time is less significant than methodology. If you want to compare a new Agile project against others you only need to search on "Agile" because there are no old Agile projects as yet. The data base itself would probably contain no Agile entries prior to 2000.

Once benchmark data is collected, it will always have value. But benchmark data collections need to be refreshed with new projects on a monthly basis. To keep track of new tools, languages, and methods monthly benchmark data for the United States should increase by about 250 projects per month in round numbers.

For specialized benchmarks such as those that deal with special topics such as embedded software or Agile development, benchmark volumes should grow at about 5 to 15 projects per month in round numbers.

The software industry does change and therefore benchmarks need to be identified by year of project completion. Table 7 illustrates approximate software quality averages from the 1960's through today with projections to 2030:

Table 7: Software Quality Levels by Decade

Era	Languages	Methods	Defect Potential	Defect Removal	Delivered Defects
1960's	Assembly	Cowboy	6.00	83%	1.02
1970's	COBOL/FORTRAN	Waterfall	5.50	85%	0.83
1980's	C, Ada	Structured	5.00	85%	0.75
1990'S	COBOL/FORTRAN	Object Oriented	4.50	87%	0.59
2000'S	Java, C#	Agile/RUP/TSP	4.25	90%	0.43
2010's	PHP/mySQL/JavaScript	Agile/RUP/TSP/Mashups	4.00	95%	0.20
2020's	Unknown	Mashups/75% reuse	3.00	98%	0.06
2030's	Unknown	SEMAT/90% reuse	2.50	99%	0.03

As can be seen from table 7 there are continuous changes in software results over time. However the software industry resembles a drunkard's walk, with about as many companies going backwards and getting worse as there are companies improving and getting better.

In fact if every company got better every year, assessments and benchmarks would probably not be needed. It is the combination of regressions and advances that make benchmarks valuable for companies and government agencies.

Executive Interest Levels in Software Benchmark Types

This article primarily deals with individual project benchmarks for software project productivity and quality levels. There are many other kinds of benchmarks as well. Table 8 shows the levels of interest in all types of benchmarks based on interviews with executives in client companies:

Table 8: Executive Interest in Software Benchmark Types

	Software Benchmark Types	CEO Interest	CFO Interest	CIO Interest	CTO Interest
1	Competitive practices within industry	10	10	10	10
2	Project failure rates (size, methods)	10	10	10	10
3	Development Schedules	10	10	10	10
4	Outsource contract success/failure	10	10	10	10
5	Project Risks	10	10	10	10
6	CMMI assessments	9	9	9	10
7	Security attacks (number, type)	10	10	10	10
8	Return on investment (ROI)	10	10	10	10
9	Total cost of ownership (TCO)	10	10	10	10
10	Customer satisfaction	10	10	10	10

11	Employee morale	10	9	10	10
12	Data quality	10	10	10	10
13	Customer support benchmarks	9	9	10	10
14	Cost of Quality (COQ)	10	10	10	10
15	Best Practices - requirements	6	9	9	10
16	Development costs	10	10	10	10
17	Team attrition rates	7	8	10	10
18	ISO standards certification	9	9	9	9
19	Best Practices - maintenance	8	9	10	10
20	Best Practices - test efficiency	8	7	9	10
21	Best Practices - defect prevention	5	6	8	10
22	Best Practices - pre-test defects	6	6	9	10
23	Technical debt	8	8	9	9
24	Productivity - project	8	8	9	10
25	Coding speed	7	8	8	8
26	Code quality (only code - nothing else)	7	7	10	10
27	Cost per defect (caution: unreliable)	6	7	10	10
28	Application sizes by type	8	8	8	9
29	Enhancement costs	8	10	10	8
30	Maintenance costs (annual)	8	9	9	9
31	Team morale	8	8	8	8
32	Litigation - patent infringements	10	10	10	10
33	Best Practices - design	5	5	9	10
34	Litigation - intellectual property	10	10	10	10
35	Portfolio maintenance costs	10	10	10	10
36	Team compensation level	10	9	9	7
37	Litigation - breach of contract	10	10	10	10
38	Earned value (EVA)	5	7	7	10
39	Methodology comparisons	5	6	9	10
40	Test coverage benchmarks	5	6	9	8
41	Productivity -activity	4	6	8	10
42	Application types	7	7	7	9
43	Litigation - employment contracts	7	8	8	8
44	Tool suites used	4	4	7	10
45	CMMI levels within industries	6	5	5	9
46	Standards benchmarks	5	5	6	8
47	Country productivity	8	5	8	9
48	Data base size	9	8	8	9
49	Industry productivity	10	7	6	9
50	Metrics used	3	3	7	9
51	Programming Languages	3	3	6	8
52	Application class by taxonomy	4	5	6	8
53	Certification benchmarks	3	3	6	7
54	Cyclomatic complexity benchmarks	1	1	5	7
55	SNAP non functional size metrics	3	4	5	8

TOTALS	412	421	475	513
	7.49	**7.65**	**8.64**	**9.33**

As can be seen all levels and types of executive are interested in benchmarks. However chief technology officers have the widest range of interest, as might be expected from the nature of CTO work.

CEO's are more interested in competitive and large-scale benchmarks than in specific project benchmarks, with the important exception of very high interest levels in applications that might be of strategic importance or cost more than $10,000,000.

Software Benchmark Metrics Used Circa 2017

Function points using the method of the International Function Point Users Group (IFPUG) are the dominant metric for software benchmarks, but far from being the only metric.

Table 9 gives a rough approximation of the probable numbers of software projects that have been benchmarked in a variety of software metrics.

It is curious sociological phenomenon that the software industry has developed at least half a dozen "clones" of IFPUG function points without any solid justification for doing so. In a sense IFPUG function points are like the QWERTY keyboard: not perfect but so widely used that alternatives seldom make much headway. It is also a curious sociological phenomenon that most of the IFPUG clones produce larger results than the original IFPUG function point.

Table 9 includes estimated numbers of benchmarks from the United States, Europe and the UK, the Pacific Rim, the Middle East, and both Central and South America. However about 90% of all benchmarks to date are from the U.S. and Europe.

Table 9: Estimated Benchmarks by Metric Circa 2017

	Benchmark Metrics	Nominal Size	Percent of IFPUG size	Benchmarks as of 2016
1	IFPUG function points	1,000	100%	40,000
2	Backfired IFPUG function points	1,000	100%	10,000
3	Physical lines of code	85,000	8500%	7,500
4	Logical code statements	50,000	5000%	5,000
5	COSMIC function points	1,143	114%	3,500
6	Mark II function points	1,060	106%	1,000
7	NESMA function points	1,040	104%	750
8	Unadjusted function points	890	89%	750
9	FISMA function points	1,020	102%	500
10	Use case points	333	33%	350

11	Story points	556	56%	300
12	Feature points	1,000	100%	200
13	Function points light	965	97%	200
14	RICE objects	4,714	471%	150
15	Pattern-match function points	1,000	100%	125
16	Fast function points	970	97%	100
17	Full function points	1,170	117%	100
18	Legacy-mined function points	1,000	100.00%	75
19	Engineered function points	1,015	102%	50
20	SNAP non-functional metrics	235	24%	25
	TOTAL			70,675

For those who are not heavily involved with benchmarks and functional metrics, the second item in table 9 is that of "backfired function points." This phrase refers to a mathematical conversion from logical code statements to function points. This method originated in IBM in the 1970's and remains in wide use even in 2018.

The last item in table 9 refers to a new variant of IFPUG function points for counting non-functional requirements. This new metric was just published at the end of 2011 so it is still early for many completed benchmarks.

Benchmark Summary and Conclusions

Software benchmarks are useful for a number of business and technical purposes. However many companies are not very accurate with historical data. Leakage is an endemic problem of historical software productivity and quality data.

Companies that either submit data to a non-profit benchmark group such as the International Benchmark Standards Group (ISBSG) or commission an on-site benchmark study from a company such as Namcook Analytics LLC usually have more accurate data than companies that just record data internally for private use.

The new parametric method of software benchmark analysis that combines parametric estimates with remote data collection provides a good combination of low cost, rapid speed, and very good accuracy levels. The method can also be used in conjunction with other forms of data collection approaches such as those of ISBSG and those of the SEI.

Readings and References on Software Benchmarks

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

Galorath, Dan; Software Sizing, Estimating, and Risk Management: When Performance is Measured Performance Improves; Auerbach Publishing, Philadelphia; 2006; ISBN 10: 0849335930; 576 pages.

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.

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; Catalog of Software Benchmark Sources; Namcook Analytics LLC, Narragansett, RI; Version 20.0; May 2012.

Jones, Capers; "Early Sizing and Early Risk Analysis"; Capers Jones & Associates LLC; Narragansett, RI; July 2011.

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).

Kan, Stephen H.; Metrics and Models in Software Quality Engineering, 2nd edition; Addison Wesley Longman, Boston, MA; ISBN 0-201-72915-6; 2003; 528 pages.

Royce, Walker; Software Project Management – A Unified Framework; Addison Wesley, Boston, MA; ISBN 0-201-30958-0; 1999; 406 pages