USING FUNCTION POINT METRICS

TO ANALYZE THE TOOLS OF SOFTWARE ENGINEERING

June 4, 2018 Version 7.0

Abstract

From data collected by the author during software assessment and benchmark studies there are major differences in the patterns of software tool usage between "leading" and "lagging" enterprises. Leading enterprises are defined as those in the top quartile of the companies evaluated in terms of software productivity, schedule adherence, and quality results. Lagging enterprises are those in lower quartile.

The most significant differences noted between laggards and leaders are in the areas of project management tools, quality assurance tools, and testing tools. Leaders tend to exceed laggards by a ratio of about 15 to 1 in the volumes of tools associated with project management and quality control. The function point metric is proving to be a useful analytical tool for evaluating the capacities of software tool suites.

Function point metrics provide a very useful tool for comparing and evaluating software tools in all categories. In this paper software tools are categorized as follows:

Project Management Tools

These are tools aimed at the software management community. These tools are often concerned with predicting the costs, schedules, and quality levels prior to development of software projects and also collecting historical data from completed projects.

Software Engineering Tools

The set of software engineering tools are those used by programmers or software engineering personnel. There are many tools in this family, and they cover a variety of activities commencing with requirements analysis and proceeding through design, coding, change control, static analysis, and personal testing such as unit test.

Software Maintenance Engineering Tools

The tools in this family are aimed at stretching out the lives of aging legacy software applications. These tools are concerned with topics such as reverse engineering, code restructuring, defect tracking, reengineering, and other activities that center on existing applications. More modern "maintenance workbenches" provide support for full renovation of legacy applications.

Software Quality Assurance Tools

The tools in the software quality assurance (SQA) set are aimed at defect prediction, prevention, defect tracking, and the other "traditional" activities of SQA teams within major corporations.

Software Testing and Static Analysis Tools

The family of testing tools has been expanding rapidly, and the vendors in this family have been on a wave of mergers and acquisitions. The test tool market place is expanding fairly rapidly, and new tools are being marketed at an increasing pace. New kinds of tools such as automated test tools and static analysis tools have joined the traditional family of test tools.

Software Documentation Tools

Every software project requires some kind of documentation support, in terms of user's guides, reference manuals, HELP text, and other printed matter. The more sophisticated software projects have a substantial volume of graphics and illustrations too, and may also use hypertext links to ease transitions from topic to topic.

Various kinds of software engineering and project management tools are used by all software professionals between 2 and 7 hours per day, every day. Because so much of the work of modern software engineering involves using tools, the usage patterns and effectiveness of tools needs greater study than the software engineering literature has provided thus far.

Capers Jones, VP and CTO, Namcook Analytics LLC
Web: www.Namcook.com
Email: Capers.Jones3@gmail.com

INTRODUCTION

There are hundreds or even thousands of commercial tools available for software development, software project management, maintenance, testing, quality control and other key activities associated with software projects. There are also hundreds of proprietary, internal tools which companies build for their own use but not for sale to others.

Every single working day software engineers, project managers, testers, and other software professionals make use of tools between 2 hours and 7 hours per day, every day. In fact without using various kinds of tools software engineering would not even exist as an occupation.

However the software literature has provided very little in the way of quantitative analysis about either usage patterns or the overall impact of tools on software productivity and quality. This report is an attempt to bring the issue of tool usage to the attention of software engineering and project management researchers.

Many commercial software tool vendors make advertising claims about the power of their tools in terms of increasing software development productivity, quality, or shortening schedules. Many of these claims are not supported by empirical data, and most appear to be exaggerated in greater or lesser degree. Indeed, the exaggerations by tool vendors did much to discredit the value of Computer Aided Software Engineering (CASE) which tended to promise more than it performed.

Considering the importance of software to business and industry, it is surprising that the topic of software tool usage has been under-reported in the software literature. Indeed, since about 1990 much of the software engineering literature has been devoted to the subject of "software process improvement" and tools have been regarded as a minor background issue. Tool alone do not distinguish between leaders and laggards, but tool usage is a significant supplemental factor.

The author's current company, Namcook Analytics LLC, performs both qualitative assessments and quantitative benchmark studies for clients. A part of the analysis is collecting data on the numbers and kinds of tools utilized for software development, project management, and other related activities.

In addition, we also record data on software productivity, quality, schedules, and other quantitative aspects of software performance as well as qualitative data on the methods and processes utilized. As of 2016 the total number of software projects in our knowledge base is rapidly pushing past 26,000 and the total number of client organizations from which we have collected data is approaching 600 companies and some government agencies.

Table 1 shows the variety of tools used by one of the author's clients on a large financial application of about 5000 function points in size. Table 1 shows the many different kinds of tools used circa 2016:

Table 1: Tools used for Finance Software Project

	Tasks	Tools Utilized
1	Architecture	**QEMU**
2	Automated test	**HP QuickTest Professional**
3	Benchmarks	**ISBSG, Namcook, Q/P Mgt Group, Davids**
4	Coding	**Eclipse, Slickedit**
5	Configuration	**Perforce**
6	Cost estimate	**Software Risk Master (SRM), SEER, SLIM**
7	Cost tracking	**Automated project office (APO), MS Project**
8	Cyclomatic	**BattleMap**
9	Debugging	**GHS probe**
10	Defect tracking	**Bugzilla**
11	Design	**Projects Unlimited**
12	Earned value	**DelTek Cobra**
13	ERP	**Microsoft Dynamics**
14	Function points 1	**Software Risk Master (SRM)**
15	Function points 2	**Function point workbench**
16	Function points 3	**CAST automated function points**
17	Graphics design	**Visio**
18	Inspections	**SlickEdit**
19	Integration	**Apache Camel**
20	ISO tools	**ISOXpress**
21	Maintenance	**Mpulse**
22	Manual test	**DevTest**
23	Milestone track	**KIDASA Software Milestone Professional**
24	Progress track	**Jira, Automated project office (APO)**
25	Project mgt.	**Automated project office (APO)**
26	Quality estimate	**Software Risk Master (SRM)**
27	Requirements	**Rational Doors**
28	Risk analysis	**Software Risk Master (SRM)**
29	Source code size 1	**Software Risk Master (SRM)**
30	Source code size 2	**Unified code counter (UCC)**
31	SQA	**NASA Goddard ARM tool**
32	Static analysis	**Kiuwan**
33	Support	**Zendesk**
34	Test coverage	**Software Verify suite**
35	Test library	**DevTest**
36	Value analysis	**Excel and Value Stream Tracking**

As can be seen tool usage circa 2018 is complex and involves dozens of tool vendors.

In analyzing this data, we perform multiple regression studies on the factors that influence the outcomes of software projects. Although the software development process is indeed a key issue, tools also exert a major impact. This report discusses some of the differences in the patterns of tool usage noted between "lagging" organizations and "leading" organizations. In terms of tool usage, the most significant differences between laggards and leaders are in the domains of project management tools and quality control tools.

PERFORMANCE OF LAGGING, AVERAGE, AND LEADING PROJECTS

Before discussing the impact of tools, it is useful to provide some background data on the results which we associate with lagging, average, and leading software projects. In our measurement studies we use the function point metric for data normalization, and this report assumes version 4.3 of the function point counting rules published by the International Function Point Users Group (IFPUG). Function points have substantially replaced the older "lines of code" (LOC) metric for all quantitative benchmark studies, since the LOC metric is not useful for large-scale studies involving multiple programming languages. (We have not yet incorporated the newer SNAP metric for non-functional requirements into these studies due to the lack of data and the ambiguity of the SNAP metric.) Neither do we use Story Points or Use-Case points since neither has any ISO standards and the results vary by over 400% from company to company.

Note that similar kinds of studies could be done using COSMIC function points, Finnish function points (FISMA) Netherlands function points (NESMA), or some of the other functional size variations. The size results would probably differ by about 15% from the current study. Other metrics such as story points and use-case points do not seem particularly well suited for tool analysis.

In our quantitative benchmark studies, as might be expected, the majority of projects are "average" in terms of productivity, quality, and schedule results. What this report concentrates on are the extreme ends of the data we collect: the outlying projects that are either much better than average or much worse than average. There are more insights to be gained by analysis of the far ends of the spectrum than by examining the projects that cluster around the center.

Let us consider what it means for a software project to be considered "average" or "leading" or "lagging" in quantitative terms. Although many attributes can be included, in this short report only six key factors will be discussed:

1. The length of software development schedules
2. Productivity rates expressed in function points per staff month
3. Defect potentials expressed in function points
4. Defect removal efficiency levels
5. Delivered defect levels
6. Rank on the capability maturity model (CMMI) of the Software Engineering Institute

In general, the set of leading companies are better in all of these factors than either the average or lagging groups. That is, their schedules are shorter, their quality levels are better, and they place higher on the SEI CMMI.

Average Software Projects

Because schedules vary with project size, the development schedules of average software projects can be approximated by raising the function point total of the project to the 0.4 power. This calculation yields the approximate number of calendar months for development between start of requirements and delivery to clients. Thus for a project of 1000 function points, raising that size to the 0.4 power yields a development schedule from start of requirements until deployment that would be roughly 15.8 calendar months.

The defect potential or number of possible bugs that might be found for average projects totals to about 4.50 bugs per function point. This is the sum of bugs or defects found in five deliverable artifacts: requirements, design, source code, user documents, and "bad fixes" or secondary defects introduced while fixing other defects. The cumulative defect removal efficiency before delivery to clients is about 85% to perhaps 92%, so the number of bugs still latent at the time of delivery is about 0.75 bugs per function point.

Software development productivity rates vary with the size and nature of the application, but are typically in the range of 6 to 10 function points per staff month for projects in the average zone.

Although the capability maturity model (CMMI) published by the Software Engineering Institute (SEI) is based on qualitative rather than quantitative results, the data shown here for average projects is representative of projects that are at Level 1 of the CMM, but not far from Level 2.

Leading Software Projects

Software projects in the upper quartile of our data base have shorter schedules, higher quality levels, and higher productivity rates simultaneously. This is not surprising, because the costs, effort, and time to find software defects is usually the largest cost driver and the most significant barrier to rapid development schedules.

To approximate the development schedule for projects in the upper quartile, raise the function point total of the application to the 0.35 power to generate the number of calendar months from requirements to deployment. For a sample project of 1000 function points in size, this calculation yields a result of about 11.2 calendar months from start of requirements until deployment.

The defect potential or number of possible bugs that might be found for leading projects is well below average, and runs to less than about 3.0 bugs per function point. The cumulative defect removal efficiency before delivery to clients is about 95% to 99%, so

the number of bugs still latent at the time of delivery is about 0.15 bugs per function point or even lower.

The reduced levels of defect potentials stem from better methods of defect prevention, while the elevated rates of defect removal efficiency are always due to the utilization of formal design reviews and code inspections. Testing alone is insufficient to achieve defect removal rates higher than about 90% so all of the top-ranked quality organizations utilize inspections also.

Here too the productivity rates vary with the size and nature of the application, but are typically in the range of 10 to 20 function points per staff month for projects in the upper quartile. (The maximum rate can exceed 30 function points per staff month.)

In terms of the capability maturity model (CMMI) published by the Software Engineering Institute (SEI) the data for the upper quartile shown is representative of projects that are at well into Level 3 of the CMMI, or higher.

Lagging Software Projects

Software projects in the lower quartile of our data base are troublesome and there is also a known bias in our data. Many projects that would be in the lower quartile if the project went all the way to completion are cancelled, and hence not studied in any depth. Therefore the projects discussed here are those which were completed, but which were well below average in results.

The effect of this situation is to make the lagging projects, as bad as they are, look somewhat better than would be the case if all of the cancelled projects were included in the same set. Unfortunately in our consulting work we are seldom asked to analyze projects that have been terminated due to excessive cost and schedule overruns. We are often aware of these projects, but our clients do not ask to have the projects included in the assessment and benchmark studies that they commission us to perform.

To approximate the development schedule for projects in the lower quartile, raise the function point total of the application to the 0.45 power to generate the number of calendar months from requirements to deployment. For a sample project of 1000 function points in size, this calculation yields a result of about 22.4 calendar months.

The defect potential or number of possible bugs that are found for lagging projects is well above average, and runs to more than about 7.0 bugs per function point. The cumulative defect removal efficiency before delivery to clients is only about 75%, so the number of bugs still latent at the time of delivery is an alarming 1.75 bugs per function point. Needless to say, lagging projects have severe quality problems, unhappy users, and horrendous maintenance expenses.

As will be discussed later, the lagging projects usually have no quality assurance tools or software quality assurance teams, and may also be careless and perfunctory in testing as well.

For laggards too the productivity rates vary with the size and nature of the application, but are typically in the range of 1.0 to 5.0 function points per staff month, although some projects in the lower quartile achieve only a fraction of a function point per staff month. (The minimum rate we've measured is 0.13 function points per staff month. The best results from the laggard group seldom approach 10 function points per staff month.)

In terms of the capability maturity model (CMMI) published by the Software Engineering Institute (SEI) the data for the lower quartile is representative of projects that are at well back at the rear of Level 1 of the CMM.

NOTE: For additional information on U.S. national averages and ranges for software schedules, productivity, and quality levels refer to the author's book Applied Software Measurement, 3rd edition, (Jones 2008) and also the newer book Software Engineering Best Practices (Jones 2009)..

A TAXONOMY OF SOFTWARE TOOL CLASSES

This report is concerned with fairly specialized tools which support software projects in specific ways. There are of course scores of general-purpose tools used by millions of knowledge workers such as word processors, spreadsheets, data bases, and the like. These general-purpose tools are important, but are not covered in the following report in depth because they are not really aimed at the unique needs of software projects.

Because tool usage is under-reported in the software literature there is no general taxonomy for discussing the full range of tools which can be applied to software projects or are deployed within software organizations. In this report, the author has developed the following taxonomy for discussing software-related tools:

Project Management Tools

These are tools aimed at the software management community. These tools are often concerned with predicting the costs, schedules, and quality levels prior to development of software projects. The set of management tools also includes tools for measurement and tracking, budgeting, and other managerial activities that are performed while software projects are underway. In other words some project management tools perform estimates and point to the future; others measure results and point to the past. Both kinds are needed.

Note that there are a number of tools available for personnel functions such as appraisals. However, these are generic tools and not aimed specifically at project management or control of software projects themselves and hence are not dealt with in this report. There

are also payroll and benefits accumulation tools, but these deal with personnel topics and not with software engineering so they are not included either.

Software Engineering Tools

The set of software engineering tools are those used by programmers or software engineering personnel. There are many tools in this family, and they cover a variety of activities commencing with requirements analysis and proceeding through design, coding, change control, static analysis, and personal testing such as unit test.

Examples of the tools in the software engineering family include design tools, compilers, assemblers, and the gamut of features now available under the term "programming support environment."

Numerically there are more vendors and more kinds of tools within the software engineering family than any of the other families of tools discussed in this report. The software engineering tools family has several hundred vendors and several thousand projects in the United States alone, and similar numbers in Western Europe. Significant numbers of tools and tool vendors also occur in the Pacific Rim and South America.

Software Maintenance Engineering Tools

The tools in this family are aimed at stretching out the lives of aging legacy software applications. These tools are concerned with topics such as reverse engineering, code restructuring, defect tracking, reengineering, and other activities that center on existing applications. More modern "maintenance workbenches" provide support for full renovation of legacy applications.

Although the family of maintenance tools is increasing, it has been an interesting phenomenon that maintenance tools have never been as plentiful nor as well marketed as software development tools.

The impact of two massive maintenance problems, the year 2000 and the Euro-currency conversion, triggered a burst of new maintenance tools circa 1995 1999. For perhaps the first time in software's history the topic of maintenance began to achieve a level of importance equal to new development.

Software Quality Assurance Tools

The tools in the software quality assurance (SQA) set are aimed at defect prediction, prevention, defect tracking, and the other "traditional" activities of SQA teams within major corporations.

It is an unfortunate aspect of the software industry that the family of quality-related tools was small during the formative years of the software occupation, during the 1960's and 1970's. In recent years the numbers of quality-related tools have been increasing fairly

rapidly, although Software Quality Assurance (SQA) tools are still found primarily only in large and sophisticated corporations. Incidentally, as a class, software quality groups are often understaffed and underfunded.

Software Testing and Static Analysis Tools

The family of testing tools has been expanding rapidly, and the vendors in this family have been on a wave of mergers and acquisitions. The test tool market place is expanding fairly rapidly, and new tools are being marketed at an increasing pace. New kinds of tools such as automated test tools and static analysis tools have joined the traditional family of test tools.

The test tool community is logically related to the software quality assurance community, but the two groups are not identical in their job functions nor in the tools which are often utilized, although there are of course duplications of tools between the two job categories.

In recent years a new class of tools called "static analysis tool" have joined traditional testing tools, although usually marketed by companies outside of traditional testing. Static analysis tools have a high efficiency in finding bugs, and are cost-effective when run prior to testing. In fact some of the structural defects found by static analysis tools are very difficult to find using normal testing tools and methods.

A wave of mergers and acquisitions has been sweeping through the test and quality tool domain. As a result, test and quality assurance tools are now starting to be marketed by larger corporations than was formerly the case, which may increase sales volumes. For many years, test and quality assurance tools were developed and marketed by companies that tended to be small and undercapitalized.

Software Documentation Tools

Every software project requires some kind of documentation support, in terms of user's guides, reference manuals, HELP text, and other printed matter. The more sophisticated software projects have a substantial volume of graphics and illustrations too, and may also use hypertext links to ease transitions from topic to topic.

For modern applications offered in the form of "software as a service" (SaaS) or web-enabled office suites such as those by Google, all of the documentation is now on line and available primarily from web sites.

The topic of documentation tools is undergoing profound changes under the impact of the word wide web and the Internet. Also the topic of work-flow management and newer technologies such as HTML, web authoring tools, and hypertext links are beginning to expand the world of documentation from "words on paper" to a rich multi-media experience where on-line information may finally achieve the long-delayed prominence which has been talked about for almost 50 years.

TOOL USAGE ON AVERAGE, LAGGING, AND LEADING PROJECTS

This section of the report discusses the ranges and variations of tools noted on lagging, average, and leading projects. Three primary kinds of information are reported in this section:

1. Variances in the numbers of tools used in lagging and leading projects
2. Variances in the function point totals of the lagging and leading tool suites
3. Variances in the daily hours of usage by software engineers and managers

The count of the numbers of tools is simply based on assessment and benchmark results and our interviews with project personnel. Although projects vary, of course, deriving the counts of tools is reasonably easy to perform.

The sizes of the tools expressed in function points are more difficult to arrive at, and have a larger margin of error. For some kinds of tools such as cost estimating tools actual sizes are known in both function point and lines of code form because the author's company builds such tools.

For many tools, however, the size data is only approximate and is derived either from "backfiring" which is conversion from lines of code to function points; or from analogy with tools of known sizes. The size ranges for tools in this report are interesting, but not particularly accurate. The purpose of including the function point size data is to examine the utilization of tool features in lagging and leading projects.

In general, the lagging projects depend to a surprising degree on manual methods and have rather sparse tool usage in every category except software engineering, where there are comparatively small differences between the laggards and the leaders.

Project Management Tools on Lagging and Leading Projects

The differences in project management tool usage are both significant and striking. The lagging projects typically utilize only three general kinds of project management tools, while the leading projects utilize 18. Indeed, the project management tool family is one of the key differentiating factors between lagging and leading projects.

In general, the managers on the lagging projects typically use manual methods for estimating project outcomes, although quite a few may use schedule planning tools such as Microsoft Project. However, project managers on lagging projects tend to be less experienced in the use of planning tools and to utilize fewer of the available features. The sparseness of project management tools does much to explain why so many lagging software projects tend to run late, to exceed their budgets, or to behave in more or less unpredictable fashions. Table 2 shows project management tool ranges:

Table 2: Numbers and Size Ranges of Software Project Management Tools
(Tool sizes are expressed in terms of IFPUG function points, version 4.3)

	Project Management Tools	Lagging	Average	Leading
1	Project planning	1,000	1,250	3,000
2	Project cost estimating			3,000
3	Statistical analysis			3,000
4	Methodology management		750	3,000
5	Reusable feature analysis			2,000
6	Quality estimation			2,000
7	Assessment support		500	2,000
8	Project office support		500	2,000
9	Project measurement			1,750
10	Portfolio analysis			1,500
11	Risk analysis			1,500
12	Resource tracking	300	750	1,500
13	Governance tools			1,500
14	Value analysis		350	1,250
15	Cost variance reporting	500	500	1,000
16	Personnel support	500	500	750
17	Milestone tracking		250	750
18	Budget support		250	750
19	Function point analysis		250	750
20	Backfiring: LOC to FP			300
21	Earned value analysis		250	300
22	Benchmark data collection			300
	Subtotal	*1,800*	*4,600*	*30,000*
	Tools	*4*	*12*	*22*

It is interesting that project managers on successful project tend to utilize tools about 4 hours per work day. On average projects the usage is about 2.5 hours per work day. On lagging projects, barely an hour per day is devoted to tool usage.

By contrast, the very significant use of project management tools on the leading projects results in one overwhelming advantage: "No surprises." The number of on-time projects in the leading set is far greater than in the lagging set, and all measurement attributes (quality, schedules, productivity, etc.) are also significantly better.

Differences in the software project management domain are among the most striking in terms of the huge differential of tool usage between the laggards and leaders. Variances in the number of tools deployed is about 7 to 1 between the leaders and the laggards, while variances in the tool capacities expressed in function points has a ratio of approximately 17 to 1 between the leaders and the laggards. These differences are far greater than almost any other category of tool.

Software Engineering Tools on Lagging and Leading Projects

The set of software engineering tools deployed has the smallest variance of any tool category between the leaders and the laggard classes. In general, unless a critical mass of software engineering tools are deployed software can't be developed at all so the basic needs of the software community have built up a fairly stable pattern of software engineering tool usage.

Table 3 shows the numbers and volumes of software engineering tools deployed, but as can easily be seen the variations are surprisingly small between the lagging, average, and leading categories.

Table 3: Numbers and Size Ranges of Software Engineering Tools
(Tool sizes are expressed in terms of IFPUG function points, version 4.3)

	Software Engineering Tools	Lagging	Average	Leading
1	Compilers	3,500	3,500	3,500
2	Program generators		3,500	3,500
3	Design tools	1,000	1,500	3,000
4	Code editors	2,500	2,500	2,500
5	GUI design tools	1,500	1,500	2,500
6	Assemblers	2,000	2,000	2,000
7	Configuration control	750	1000	2,000
8	Source code control	750	1000	1,500
9	Static analysis		1000	1,500
10	Automated testing		1000	1,500
11	Data modeling	750	1000	1,500
12	Debugging tools	500	750	1,250
13	Data base design	750	750	1,250
14	Capture/playback	500	500	750
15	Library browsers	500	500	750
16	Reusable code analysis			750
	Subtotal	*15,000*	*22,000*	*29,750*
	Tools	*12*	*14*	*16*

Software engineers are the most intense occupation in terms of tool usage. There is comparatively little difference between lagging, average, and leading projects in terms of daily hours of tool usage: somewhere between 5 and 9 hours per working day, every day.

There are some differences in software engineering tool usage, of course, but the differences are very minor compared to the much more striking differences in the project management and quality assurance categories.

The overall features and sizes of software engineering tools have been increasing as tool vendors add more capabilities. About 10 years ago when the author first started applying function point metrics to software tools, no software engineering tools were larger than 1000 function points in size, and the total volume of function points even among the

leading set was only about 10,000 function points. A case can be made that the power or features of software engineering tools have tripled over the last 10 years.

As can be seen from table 3, although there are some minor differences in the tool capacities between the leaders and the laggards, the differences in the number of software engineering tools deployed is almost nonexistent.

A very common pattern noted among assessment and benchmark studies is for the software development teams and tool suites to be fairly strong, but the project management and quality tool suites to be fairly weak. This pattern is often responsible for major software disasters, such as the long delay in opening up the Denver Airport because the luggage-handling software was too buggy to be put into full production.

Software Maintenance Engineering Tools on Lagging and Leading Projects

When the focus changes from development to maintenance (defined here as the combination of fixing bugs and making minor functional enhancements) the tool differentials between the leaders and the laggards are much more significant than for development software engineering.

For many years, software maintenance has been severely understated in the software literature, and severely underequipped in the tool markets. Starting about 10 years ago the numbers of software personnel working on aging legacy applications began to approach and in some cases exceed the numbers of personnel working on brand new applications. This phenomenon brought about a useful but belated expansion in software maintenance tool suites. Table 4 shows the variations in software maintenance engineering tools:

Table 4: Numbers and Size Ranges of Maintenance Engineering Tools
(Tool sizes are expressed in terms of IFPUG function points, version 4.3)

	Maintenance Engineering Tools	Lagging	Average	Leading
1	Maintenance work benches		1,500	3,500
2	Reverse engineering		1,000	3,000
3	Reengineering		1,250	3,000
4	ITIL support tools		1,000	3,000
5	Configuration control	500	1,000	2,000
6	Code restructuring			1,500
7	Customer support		750	1,250
8	Debugging tools	750	750	1,250
9	Defect tracking	500	750	1,000
10	Complexity analysis			1,000
11	Error-prone module analysis		500	1,000
12	Incident management	250	500	1,000
13	Reusable code analysis		500	750
	Subtotal	*1,750*	*9,500*	*23,250*
	Tools	4	11	13

Personnel on leading software maintenance projects tend to use tools more than 4 hours per day. Laggards use tools less than 2 hours per day, while personnel on average projects use tools about 3 hours per day.

As the overall personnel balance began to shift from new development to maintenance, software tool vendors began to wake up to the fact that a potential market was not being tapped to the fullest degree possible.

The differences between the leaders and the laggards in the maintenance domain are fairly striking and include about a 4 to 1 differential in numbers of tools deployed, and a 13 to 1 differential in the function point volumes of tools between the leaders and the laggards.

The emergence of two massive business problems had a severe impact on maintenance tools, and on maintenance personnel as well. The year 2000 software problem and the ill-timed Euro-currency conversion work both triggered major increases in software maintenance tools that can deal with these specialized issues.

Other issues that affect maintenance work include the use of COTS packages, the use of open-source applications, and the emergence of Software as a Service. Also the Information Technology Infrastructure Library (ITIL) has also impacted both maintenance and customer support tools.

Between about 2000 and 2016 industry maintenance “leaders” tend to have almost twice the maintenance tool capacities as those available prior the elevation of maintenance to a major occupation.

Software Quality Assurance Tools on Lagging and Leading Projects

When the software quality assurance tool suites are examined, one of the most striking differences of all springs into focus. Essentially the projects and companies in the "laggard" set have either no software quality assurance function at all or no more than 1 kind of tool in use, as can be seen in table 5.

Table 5: Numbers and Size Ranges of Software Quality Assurance Tools
(Tool sizes are expressed in terms of IFPUG function points, version 4.3)

	Quality Assurance Tools	Lagging	Average	Leading
1	Quality estimation tools			2,000
2	Quality measurement tools		750	1,500
3	Six-sigma analysis			1,250
4	Data quality analysis			1,250
5	QFD support			1,000
6	TQM support			1,000
7	Inspection support			1,000
8	Reliability estimation			1,000
9	Defect tracking	250	750	1,000
10	Complexity analysis		500	1,000
	Subtotal	*250*	*1,500*	*12,000*
	Tools	1	3	*10*

Quality assurance provides the greatest contrast between tool usage and lack of tool usage. Quality assurance personnel on leading projects use tools almost 5 hours per day; only about 2 hours per day on average projects. For lagging projects, tools might not even be used. If they are, usage is seldom more than 1 hour per day.

By contrast, the leaders in terms of delivery, schedule control, and quality all have well-formed independent software quality assurance groups that are supported by powerful and growing tool suites.

Unfortunately, even leading companies are sometimes understaffed and underequipped with software quality assurance tools. In part, this is due to the fact that so few companies have software measurement and metrics programs in place that the significant business value of achieving high levels of software quality is often unknown to the management and executive community.

Several tools in the quality category are identified only by their initials, and need to have their purpose explained. The set of tools identified as "QFD support" are those which support the special graphics and data analytic methods of the "quality function deployment" methodology.

The set of tools identified as "TQM support" are those which support the reporting and data collection criteria of the "total quality management" methodology.

The other tools associated with the leaders are the tools of the trade of the software quality community: tools for tracking defects, tools to support design and code inspections, quality estimation tools, reliability modeling tools, and complexity analysis tools.

Complexity analysis tools are fairly common, but their usage is much more frequent among the set of leading projects than among either average or lagging projects. Complexity analysis is a good starting point prior to beginning complex maintenance work such as error-prone module removal .

Another good precursor tool class prior to starting major maintenance tasks would be run static analysis tools on the entire legacy application. However a caveat is that static analysis tools only support about 25 languages out of the approximate 2,500 programming languages in existence. Static analysis is available for common languages such as C, C++, C#, Java, COBOL, FORTRAN, and some others. Static analysis is not available for the less common languages used for legacy applications such as JOVIAL, CMS2, CHILL, or CORAL.

Unfortunately, since the laggards tend to have no quality assurance tools at all, the use of ratios is not valid in this situation. In one sense, it can be said that the leading projects have infinitely more software quality tools than the laggards, but this is simply because the lagging set often have zero quality tools deployed.

Software Testing Tools on Lagging and Leading Projects

Although there are significant differences between the leading and lagging projects in terms of testing tools, even the laggards test their software and hence have some testing tools available.

Note that there is some overlap in the tools used for testing and the tools used for quality assurance. For example, both test teams and software quality assurance teams may both utilize complexity analysis tools.

Incidentally, testing by itself has never been fully sufficient to achieve defect removal efficiency levels in the high 90% range. All of the “best in class” software quality organizations use a synergistic combination of requirements inspections, static analysis, design and code inspections, and multiple testing stages. This combined approach can lead to defect removal efficiency levels that may top 99% in best case situations, and always top the current U.S. average of 85% or so.

Table 6: Numbers and Size Ranges of Software Testing Tools
(Tool sizes are expressed in terms of IFPUG function points, version 4.3)

	Testing Tools	Lagging	Average	Leading
1	Test case generation			1,500
2	Automated test tools			1,500
3	Complexity analysis		500	1,500
4	Static analysis tools		500	1,500
5	Data quality analysis			1,250
6	Defect tracking	500	750	1,000
7	Test library control	250	750	1,000
8	Performance monitors		750	1,000
9	Capture/playback		500	750
10	Test path coverage			350
11	Test case execution			350
	Subtotal	*750*	*3,750*	*11,700*
	Tools	3	6	11

Modern testing is highly dependent on tool usage. On leading projects test tools are used almost 7 hours per business day; about 5 hours on average projects; and perhaps 4 hours per day even on lagging projects. Testing circa 2010 is intensely automated.

The differences in numbers of test tools deployed ranges by about 3.5 to 1 between the leading and lagging projects. However, the tool capacities vary even more widely, and the range of tool volumes is roughly 16 to 1 between the leaders and the laggards.

This is one of the more interesting differentials because all software projects are tested and yet there are still major variations in numbers of test tools used and test tool capacities. The leaders tend to employ full-time and well-equipped testing specialists while the laggards tend to assign testing to development personnel, who are often poorly trained and poorly equipped for this important activity.

For a more extensive discussion of the differences between leaders and laggards in terms of both quality assurance and testing refer to the author's book Software Quality - Analysis and Guidelines for Success (Jones 1996) and the more recent Software Engineering Best Practices (Jones 2009).

These books also discuss variations in the numbers and kinds of testing activities performed, and also variations in the use of defect tracking tools, use of formal design and code inspections, quality estimation, quality measurements, and many other differentiating factors.

Unfortunately, none of the major vendors of test tools and only a few of the vendors of quality assurance tools have any empirical data on software quality or provide information on defect removal efficiency levels. The subject of how many bugs can actually be found by various kinds of review, inspection, and test is the most important single topic in the test and quality domain, but the only published data on defect removal

tends to come from software measurement and benchmark companies rather than from test tool and quality tool companies.

Software Documentation Tools on Lagging and Leading Projects

Almost all software projects require documentation, but very few are documented extremely well. The set of documentation tools is undergoing profound transformation as on-line publishing and the world wide web begin to supplant conventional paper documents.

Note that some of these tools included here in the documentation section are also used for requirements, specifications, plans, and other documents throughout the software development cycle. For example, almost every knowledge worker today makes use of "word processing" tools so these tools are not restricted only to the software documentation domain.

As on-line publishing grows, this category is interesting in that the "average" and "leading" categories are fairly close together in terms of document tool usage. However the laggards are still quite far behind in terms of both numbers of tools and overall capacities deployed.

Table 7: Numbers and Size Ranges of Software Documentation Tools
(Tool sizes are expressed in terms of IFPUG function points, version 4.3)

	Documentation Support Tools	**Lagging**	**Average**	**Leading**
1	Word processing	3,000	3,000	3,000
2	Web publishing	2000	2,500	3,000
3	Desktop publishing	2,500	2,500	2,500
4	Graphics support	500	500	2,500
5	Multimedia support		750	2,000
6	Grammar checking			500
7	Dictionary/thesaurus	500	500	500
8	Hypertext support		250	500
9	Web publishing	200	400	500
10	Scanning			300
11	Spell checking	200	200	200
	Subtotal	*8,900*	*10,600*	*15,500*
	Tools	7	9	11

There is not a great deal of difference in tool usage among writers and illustrators as of 2018. All three projects, lagging, average, and leading, tend to use tools between 4 and 6 hours per business day.

As web publishing becomes more common, it is likely that conventional paper documents will gradually be supplanted by on-line documents. The advent of the "paperless office" has been predicted for years but stumbled due to the high costs of storage.

Now that optical storage is exploding in capacities and declining in costs, optical on-line storage is now substantially cheaper than paper storage, so the balance is beginning to shift towards on-line documentation and the associated tool suites.

In the documentation domain the variance between the leaders and the laggards is 1.5 to 1 in the number of tools deployed, and almost 2 to 1 in the volumes of tools deployed. The differences in the documentation category are interesting, but not so wide as the differentials for project management and quality assurance tools.

Overall Tool Differences between Laggards and Leaders

To summarize this analysis of software tool differentials between lagging and leading organizations, table 8 shows the overall numbers of tools noted in our assessment and benchmark studies. Table 8 shows the largest ratios between leaders and laggards at the top:

Table 8: Ratios Between Tools Used by Lagging Project Compared to Leading Projects

TOTAL TOOLS UTILIZED	Lagging	Average	Leading	Ratios
Quality Assurance Tools	1	3	10	1 to 10.0
Project Management Tools	4	12	22	1 to 5.5
Testing Tools	3	6	11	1 to 3.6
Maintenance tools	4	11	13	1 to 2.6
Testing Tools	3	6	11	1 to 3.6
Documentation Support Tools	7	9	11	1 to 1.4
TOTAL	31	55	83	1 to 2.5

As can be seen, there is roughly a 2.5 to 1 differential in the numbers of tools deployed on leading projects as opposed to the numbers of tools on lagging projects. The major differences are in the project management and quality assurance tools, where the leaders are very well equipped indeed and the laggards are almost exclusively manual and lack most of the effective tools for both project management and quality control purposes.

When tool capacities are considered, the range of difference between the lagging and leading sets of tools is even more striking and the range between leaders and laggards jumps up to about a 4.3 to 1 ratio.

The use of function point totals to evaluate tool capacities is an experimental method with a high margin of error, but the results are interesting. Although not discussed in this report, the author's long range studies over a 10 year period has found a substantial increase in the numbers of function points in all tool categories.

It is not completely clear if the increase in functionality is because of useful new features, or merely reflects the "bloat" which has become so common in the software world. For

selected categories of tools such as compilers and programming environments, many of the new features appear to be beneficial and quite useful.

The added features in many project management tools such as cost estimating tools, methodology management tools, and project planning tools are also often giving valuable new capabilities which were long needed.

For other kinds of tools however, such as word processing, at least some of the new features are of more questionable utility and appear to have been added for marketing rather than usability purposes.

Table 9 illustrates the overall differences in tool capacities using function point metrics as the basis of the comparison. Table 8 shows the largest differences in function point tool usage at the top:

Table 9: Ratio of Tool Features Used by Lagging Project Compared to Leading Projects
(Tool sizes are expressed in terms of IFPUG function points, version 4.3)

TOTAL TOOL FUNCTION POINTS	Lagging	Average	Leading	Ratios
Quality Assurance Tools	250	1,500	12,000	1 to 48.0
Project Management Tools	1,800	4,600	30,000	1 to 16.6
Testing Tools	750	3,750	11,700	1 to 15.6
Maintenance Engineering Tools	1,750	9,500	23,250	1 to 13.2
Software Engineering Tools	15,000	22,000	29,750	1 to 1.9
Documentation Support Tools	8,900	10,600	15,500	1 to 1.7
TOTAL	28,200	51,950	122,200	1 to 4.3

It is painfully obvious that lagging projects use inaccurate manual estimates and are very poor in software quality control. No wonder laggards are troubled by cancelled projects, cost overruns, lengthy schedule delays, and very bad quality after deployment.

SUMMARY AND CONCLUSIONS

Although software tools have been rapidly increasing in terms of numbers and features, the emphasis on software process improvement in the software engineering literature has slowed down research on software tool usage.

Both software processes and software tools have significant roles to play in software engineering, and a better balance is needed in research studies that can demonstrate the value of both tools and process activities.

The use of function point metrics for exploring software tool capacities is somewhat experimental, but the results to date have been interesting and this method may well prove to be useful.

Long-range analysis by the author over a 10 year period using function point analysis has indicated that software tool capacities have increased substantially, by a range of about 3 to 1. It is not yet obvious that the expansion in tool volumes has added useful features to the software engineering world, or whether the expansion simply reflects the "bloat" that has been noted in many different kinds of software applications.

However modern quality control tools such as static analysis and modern project management tools such as parametric cost estimates do add significant value in terms of improved software results.

REFERENCES AND READINGS

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.

Howard, Alan (Ed.); Software Productivity Tool Catalogs; (in seven volumes); Applied Computer Research (ACR; Phoenix, AZ; 1997; 300 pages.

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.

Jones, Capers; Function Point Metrics and Software Usage Patterns; Capers Jones & Associates; Narragansett, RI; Dec. 1, 2009.

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, 3rd edition 2008; ISBN 978-0-07-150244-3; 668 pages; 3rd edition (March 2008).

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

Jones, Capers; Software Engineering Best Practices; McGraw Hill, New York; 2009.

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

Jones, Capers; Assessment and Control of Software Risks; Prentice Hall, 1994; ISBN 0-13-741406-4; 711 pages.

Jones, Capers; Patterns of Software System Failure and Success; International Thomson Computer Press, Boston, MA; December 1995; 250 pages; ISBN 1-850-32804-8; 292 pages.

Jones, Capers; Software Quality – Analysis and Guidelines for Success; International Thomson Computer Press, Boston, MA; ISBN 1-85032-876-6; 1997; 492 pages.

Jones, Capers: “Sizing Up Software;” Scientific American Magazine, Volume 279, No. 6, December 1998; pages 104-111.

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.

Pressman, Roger; Software Engineering – A Practitioner’s Approach; McGraw Hill, NY; 6th edition, 2005; ISBN 0-07-285318-2.