Using Artificial Intelligence (AI)

For Large Software Engineering Projects – Part 1

Capers Jones

Foreword: The following pages constitute Part 1 of *Using Artificial Intelligence for Large Software Engineering Projects* by Capers Jones, released for distribution to the International Function Point Users Group (IFPUG) in January of 2025. Topics in this extract include: Function Points and a Glimpse Towards 2054, Abridged History or Software Problems, and Software Engineering Progress.

Note: Many of the illustrations in the original document were produced using artificial intelligence. As of February, 2025 AI-generated illustrations cannot be copyrighted. The remaining content of this document, while shared, is copyrighted by Capers Jones, 2025.

Acknowledgment: The IFPUG community expresses its appreciation to Mr. Jones for his lifelong pursuit of metrics-based state of and recommended improvements for software development practices globally.

Joe Schofield

Circa 2025, the U.S. average software productivity is roughly 8.00 function points per staff month or 16.5 work hours per function point. The current U.S. average software quality combines a defect potential of about 4.25 bugs per function point and defect removal efficiency (DRE) of only about 92.5%. Delivered defects average about 0.319 per function point. (Function points are better than the older "lines of code" (LOC) metric since they can measure non-coding work and measure bugs in requirements and design, which are invisible using LOC.)

The percentage of large software projects > 10,000 function points that are canceled and never delivered is close to 35%. The percentage of large software projects that exceed planned schedules and exceed their planned budgets is about 75%.

Software engineering in 2024 has more failing projects and more delays and cost overruns than any other technical industry. Interviews with CEOs of Fortune 500 companies indicate that CEOs universally regard software engineering as their least competent and least professional technical organizations due to their frequent failures and excessive overruns.

To become an effective and trusted technical profession software engineering needs to make major improvements over the next 20 years between 2024 and 2054. From examining current and advanced technologies, the author proposes these targets for software engineering in 2054:

Table 1: Proposed Software Engineering Targets for 2054 Compared to 2024

Software Engineering Factors	2024	2054	Difference	Percent Improvement
Average % of reusable components	10.00%	85.00%	75.00%	750.0%
Function points per month	8.00	96.00	88.00	1100.0%
Work hours per FP	16.50	1.38	-15.13	-91.7%
$ per function point (development)	$1,000.00	$125.00	-$875.00	-87.5%
$ per function point (maintenance)	$1,250.00	$150.00	-$1,100.00	-88.0%
Total Cost of Ownership (TCO)	$2,250.00	$275.00	-$1,975.00	-87.8%
Defect Potential per function point	4.00	1.00	-3.00	-75.0%
Defect removal efficiency (DRE)	92.50%	99.50%	7.00%	7.6%
Delivered defects per FP	0.30	0.01	-0.30	-98.3%
High-severity defects delivered	0.05	0.00	-0.04	-81.9%
Security flaws delivered	0.02	0.00	-0.02	-81.9%
High quality, on-time, on schedule	60.00%	99.00%	39.00%	65.0%
Cancelled projects	15.00%	1.00%	-14.00%	-93.3%
Cost overruns	35.00%	2.00%	-33.00%	-94.3%
Schedule delays	40.00%	2.00%	-38.00%	-95.0%
Successful cyber attacks	12.50%	0.50%	-12.00%	-96.0%
Breach of contract litigation	5.00%	0.50%	-4.50%	-90.0%

Custom designs for software applications and manual coding are intrinsically expensive, error-prone, and slow regardless of which programming languages are used and which development methodologies are used. Agile has been faster than waterfall, but it is still slow compared to actual business needs. No manual methods by human software engineers can match future development using artificial intelligence. Unfortunately, this means many software engineers may lose their jobs.

Note that reusability is not just valuable for software. It has been a major part of the industrial revolution for all industries. The approximate volume of standard reusable parts has been increasing in all industries other than software:

Products	**Approximate Reuse Percent Circa 2024**
1) Smart phones	95%
2) Personal computers	90%
3) Firearms (rifles, pistols, etc.)	85%
4) Home Construction (Japan)	75%
5) Automobiles	70%
6) Home appliances	65%
7) Telecommunications switches	60%
8) Aircraft (commercial)	50%
9) Home construction (U.S.)	15%
10) Software	**10%**

As of 2024 software seems to lag all other major products in terms of certified reusable components. Modern reusable parts can be dated to the American inventor Eli Whitney and the year 1794. Up until Whitney musket construction was a skilled craft and weapon makers could only produce one or two firearms per month. Whitney received a contract to product 10,000 muskets in only two years. Whitney's construction of fire arms from standard reusable components lowered the skill level needed for manufacture, and vastly increased both productivity and quality. These same principles are urgently needed in 2024 by the software engineering community.

The only effective solution for software engineering is to move towards construction of applications using standard reusable materials rather than custom design and development. The idea is to build software more like Ford builds automobiles on an assembly line rather than like the custom design and manual construction of a Formula 1 race car.

An ordinary passenger car and a Formula 1 race car have about the same number of mechanical parts, but the race car costs at least 10 times more to build due to the large volumes of skilled manual labor involved. The schedule would be more than 10 times longer as well. Custom designs and manual construction are intrinsically slow and expensive in every industry.

Within 10 years both race cars and regular autos for consumers will be built by artificial intelligence.

If you compare the costs and schedules of building an 80-story office building to an 80,000-function point software system, the software is much more expensive and also much slower.

When deployed the software is much less reliable and has many more defects that interfere with use than the other two. Worse, the software is much more likely to be attacked by external criminals seeking to steal data or interfere with software operation.

These problems are endemic but not impossible to cure. It is technically possible today in 2015 to build some software applications from standard reusable components. It is also possible to raise the immunity of software to external cyber-attack.

In the future more and more standard components embedded in AI software development tools will expand the set of applications that can be assembled from certified standard parts free from security vulnerabilities rather than needing custom design and laborious manual coding that tend to introduce security flaws. Assembly from certified components can be more than 10 times faster and cheaper than the best manual methods such as agile, and also much more secure than today's norms where security vulnerabilities are rampant.

Another approach to effective development of large software applications will be artificial intelligence, although AI is not yet widely used for software in 2024. A combination of AI and a large library of reusable components could improve large system development by more than an order of magnitude. Probably quality and reliability would improve as well, although it is too soon to know the quality and reliability of artificial intelligence software development.

A Brief History of Software Engineering Problems

Software engineering has been a manual and labor-intensive craft since the early days of computers in the late 1940's and early 1950's. However, the problems became worse towards the end of the 1960's when software applications began to swell past 1,000,000 lines of code or 10,000 function points. Table 2 summarizes a large number of software engineering problems between 1950 and 2024:

Table 2: A Brief History of Software Engineering from 1950 to 2024

1950	Machine language is too complex for increasing software size
1952	Low-level assembly languages released to improve on machine language
1957	Unstructured "cowboy" development is proven hazardous
1958	COBOL, ALGOL, and FORTRAN begin the market for high-level languages
1960	Structured development methods begin to replace unstructured "cowboy" development
1960	Software engineering, computer science curricula start at universities
1960	Waterfall development becomes normal for large defense software projects
1964	Defect removal becomes the # 1 software cost driver > 1000 FP

1965	Flowcharts proven inadequate; HIPO, Nassi-Shneiderman, and later UML augment designs
1969	Paperwork becomes the #2 software cost driver > 1000 FP
1970	Requirements creep becomes the #3 cost driver > 1000FP
1970	IBM has first major schedule delays on OS/360
1970	IBM discovers error-prone modules (EPM); 5% EPM contain > 50% of application defects.
1970	IBM discovers serious mathematical errors with "lines of code" metric
1971	IBM discovers > 5% of test cases have errors.
1972	Applications grow > 10,000 function points or 1,000,000 LOC; with schedule and cost overruns rampant
1972	IBM discovers that testing defect removal efficiency for normal test sequence is < 85%
1972	IBM discovers serious mathematical errors with "cost per defect" metric
1973	IBM introduces formal inspections to solve poor testing problems
1973	IBM builds first internal software parametric estimation tool (Development Planning System)
1973	IBM invests in function points to solve LOC metrics errors
1973	IBM develops "defect potential" and "defect removal efficiency" metrics to prove inspections
1975	Requirements creep tops 1% per calendar month > 1000 FP
1975	Human deaths and injuries due to software increase in frequency
1975	IBM develops "backfiring" or mathematical conversion from LOC to function points
1976	Project management becomes #5 cost driver > 1000 FP
1976	Cyclomatic complexity metric released and proves harm of complex code
1977	Defect potentials top 5.00 per function point > 1000 FP
1977	IBM attempts a universal language (PL/I) which does not replace other languages
1978	Defect removal efficiency usually below 90% for projects > 1000 FP
1978	Design errors are > 20% of all software defects
1978	IBM places function point metrics in public domain as a public service
1978	Software antitrust litigation becomes significant
1978	IBM discovers that bad-fixes in software > 7% (these are bugs in bug repairs)
1979	Coding drops to #4 cost driver > 1000 FP due to huge document and defect repair costs
1980	Requirements errors > 15% of all software defects
1980	Debugging tools released to help programmers find bugs
1981	First parametric estimation tool released to improve estimate accuracy (COCOMO)
1982	Canceled projects > 25% for projects > 10,000 function points
1982	Canceled projects > 50% for projects > 100,000 function points
1982	International Function Point Users Group (IFPUG) formed to standardize function points
1983	Schedule delays > 25% for projects > 1000 FP
1983	IBM develops Joint Application Design (JAD) to improve requirements rigor
1984	Cost overruns >20% for projects > 1000 FP
1984	First parametric tool based on function points released (SPQR/20)
1984	Static analysis tools released to improve testing inadequacy
1985	Patent violation litigation becomes significant
1985	Errors found in manual correctness proofs; automated proofs better
1985	Software Engineering Institute (SEI) formed to help solve software problems
1986	Computer hacking becomes serious
1986	First certification exam for function points offered by IFPUG
1986	Large projects found to need formal project offices or risk serious delays and overruns

1986	New programming languages developed at more than 1 per month; nobody knows why
1987	Cyber-attacks and cyber-crime become serious
1987	First version of SEI capability maturity model (CMM) released
1988	Test coverage tools arrive - show test coverage is often < 80% due to high code complexity
1991	Average defect removal efficiency of testing stages such as unit test shown to be < 35%
1991	Average defect removal efficiency of formal inspections shown to be > 80%
1992	Outsource breach of contract litigation becomes significant
1993	Cyber-crime units created by FBI, CIA, Secret Service, etc.
1994	Automated requirements models begin to add rigor to requirements definitions
1995	Coding errors drop from > 70% to < 30% of totals due to high-level languages
1997	International Software Benchmark Standards Group (ISBSG) formed to improve benchmark access
1998	Arrival of Euro causes expensive changes to thousands of applications
1998	Offshore outsourcing begins to erode U.S. and European software jobs
1999	Social networks spread personal information globally
1999	ERP deployment found to be troublesome; ERP quality is poor due to huge sizes
2000	Y2K problem damages thousands of software projects
2001	Agile invented to solve waterfall development problems
2001	Major software failures damage stock markets, air traffic, medical devices, and airline reservations
2003	Automated testing tools arrive to speed up manual testing
2006	Maintenance costs become larger than development costs in U.S. and Europe
2006	ISO standards published on functional size measures
2008	Major industrial cyber-attacks in several countries
2008	Brazil becomes 1st country to mandate function points; Italy, Japan, South Korea, Malaysia follow
2009	Average defect removal efficiency of static analysis shown to be > 55%; false positives are 5%
2010	Cyber warfare units formed in all major countries
2010	IBM coins "greenfield" for new software and "brownfield" for software impacted by legacy
2010	Globally "brownfield" projects are about 65%; "greenfield projects are about 35%
2010	Interviews show that software historical data "leaks" and is only about 37% complete
2010	Interviews show that major leakage include unpaid overtime, management, and specialists
2010	Interviews show that historical quality data "leaks" and is less than 50% complete
2010	Interviews show that desk checks, static analysis, and unit test bugs are almost never tracked
2011	Namcook files patent application on early-high speed sizing method
2011	Software Risk Master (SRM) released to speed up sizing and do earlier estimates
2011	Software Engineering Methods and Theory (SEMAT) proposed to help software problems
2011	OMG standards published on automated function point counts
2012	Automated project office (APO) released to improve tracking, visibility
2012	Hundreds of legacy applications coded in dead languages (CHILL, Coral, Bliss, Mumps, etc.)
2012	The new metric of "Technical Debt" is published and becomes popular.
2012	The new metric "SNAP" (software non-functional assessment process) created by IFPUG.
2012	Less than 50% of global projects use static analysis; less than 15% use formal inspections
2012	Average application uses about 3 languages such as Java, HTML, and SQL; makes code counts hard
2013	Automatic function point counts developed to speed up counting

2013	U.S. software costs now about 60% maintenance; 40% development
2013	Cost of Quality (COQ) now about 43% of Total Cost of Ownership (TCO)
2014	Software industry has 58 named development methods; few have empirical data of success
2014	Selecting programming languages resembles joining religious cults due to lack of empirical data
2014	Selecting development methodologies resembles joining religious cults due to lack of empirical data
2015	Number of programming languages tops 2,500; less than 50 are widely used; nobody knows why
2015	Software failures, schedule delays, and cost overruns are still rampant > 10,000 function points
2015	Average software defect potentials now about 4.00 per function points; DRE now about 92.5%
2015	Average software productivity now about 8.00 function points per month; 16.5 work hours per FP
2015	Function points now divided among COSMIC, FISMA, IFPUG, NESMA, and a dozen others
2015	Frequent cyber-attacks on U.S. government and corporate data; most from foreign governments.
2015	Inaccurate manual estimates still top 80% globally; accurate parametric estimates below 20% globally
2015	Function point metrics now #1 in Europe, South America, and U.S; but lag in China and Russia
2015	Less than 1% of software projects globally collect accurate benchmark data; less than 5% any data
2015	Inaccurate metrics, incomplete measurements, inaccurate estimates are global software problems
2016	Bug repairs now #1 cost driver for software applications
2016	Cyber-attacks and cyber-recovery approach #2 cost driver for software applications
2016	Software has 65 named development methodologies: none are suitable for all applications.
2016	Software has > 3,000 programming languages: none are suitable for all applications
2016	An average application in 2016 uses about 2.5 programming languages
2020	Corona virus arrives in U.S. and disrupts many businesses
2020	Claims of election cheating involving computerized vote counts
2021	Increasing litigation for software failures
2022	Failures of important government software application
2023	Concerns of military software due to Ukraine Russia launch failures
2023	Concerns begin about artificial intelligence replacing human jobs
2023	Artificial intelligence is approaching readiness for software development
2024	Court decisions on AI copyrights for software, books, music, etc.

As can be seen by Table 2, software engineering is still troubled by major problems even in 2024. Software remains a troubling industry plagued by far too many canceled projects, and far too many cost and schedule overruns.

Poor quality of delivered software is also an industry disgrace. Every software project should measure defect removal efficiency (DRE), and all software outsource contracts should mandate DRE levels > 97.5%. Critical applications (medical devices, weapons, finance, etc.) should mandate DRE levels above 99.5%. The current average is only about 92.50% so quality improvements are urgently needed.

One major way to improve quality is to use design and code inspections. The next picture shows a future inspection in progress in 2034 with humans and robots discussing code together: