# EVALUATING TEN SOFTWARE DEVELOPMENT METHODOLOGIES

Source
https://web.archive.org/web/20130915171920/http://namcookanalytics.com/evaluating-ten-software-development-methodologies/
Capers Jones, Vice President and CTO
Namcook Analytics LLC
Email: Capers.Jones3@Gmail.com.
Web: www.Namcook.com


---

Version 3.0 May 2, 2013

## Abstract

As this is written there are about 55 named software development methods in use, and an even larger number of hybrids. Some of the development methods include the traditional waterfall approach, various flavors of agile, the Rational Unified Process (RUP), the Team Software Process (TSP), V-Model development, Microsoft Solutions Framework, the Structured Analysis and Design Technique (SADT), Evolutionary Development (EVO), Extreme Programming (XP), PRINCE2, Merise, model-based development, and many more. Only 10 methods are evaluated here, since evaluating all 55 would take too many pages.

In general selecting a software development methodology has more in common with joining a cult than it does with making a technical decision. Many companies do not even attempt to evaluate methods, but merely adopt the most popular, which today constitute the many faces of agile.

This article uses several standard metrics including function points, defect removal efficiency (DRE), Cost of Quality (COQ), and Total Cost of Ownership (TCO) to compare a sample of contemporary software development methods.

The data itself comes from studies with a number of clients who collectively use a wide variety of software methods. The predictions use the author's proprietary Software Risk Master™ tool which can model all 55 software development methodologies.

# INTRODUCTION

The existence of more than 55 software development methods, each with loyal adherents, is a strong message that none of the 55 is capable of handling all sizes and kinds of software applications.

Some methods work best for small applications and small teams; others work well for large systems and large teams; some work well for complex embedded applications; some work

well for high-speed web development; some work well for high-security military applications. How is it possible to select the best methodology for specific projects? Is one methodology enough, or should companies utilize several based on the kinds of projects they need to develop?

Unfortunately due to lack of quantified data and comparisons among methodologies, selecting a software development method is more like joining a cult than a technical decision. Many companies do not even attempt to evaluate alternative methods, but merely adopt the most popular method of the day, whether or not it is suitable for the kinds of software they build.

When software methodologies are evaluated the results bring to mind the ancient Buddhist parable of the blind men and the elephant. Different methods have the highest speed, the highest quality, and the lowest total cost of ownership.

(In the original parable the blind man who touched the trunk thought an elephant was like a snake. The blind man who touched the side thought an elephant was like a wall. The blind man who touched the tusk thought an elephant was like a spear; the blind man who touched the tail thought an elephant was like a rope.)

## Combinations of Factors that Affect Software Projects

An ideal solution would be to evaluate a variety of methods across a variety of sizes and types of software. However that is difficult because of combinatorial complexity. Let us consider the major factors that are known to have an impact on software project results:

| Factor | Number of Possibilities |
|---|---|
| Methodologies | 55 |
| Programming languages | 50 |
| Nature, class, and type of application | 15 |
| Capability Maturity Model Levels | 5 |
| Team experience (low, average, high) | 3 |
| Size plateau of application (small, medium, large) | 3 |
| Application complexity (low, average, high) | 3 |
| Combinations of factors | 5,568,750 |

Since the number of combinations is far too large to consider every one, this article will make simplifying assumptions in order to focus primarily on the methodologies, and not on all of the other factors.

In this article the basic assumptions will be these:

| Application size | 1000 function points |
|---|---|
| Programming languages | C and C++ |
| Logical code statements | 75,000 |
| Requirements creep | Omitted |
| Deferred features | Omitted |
| Reusable features | Omitted |

| **Application size** | **1000 function points** |
|---|---|
| Team experience | Average |
| Complexity | Average |
| Cost per staff month | $7,500 |

By holding size, languages, complexity, and team experience at constant levels it is easier to examine the impacts of the methodologies themselves. There are unfortunately too many methodologies to consider all of them, so a subset of 10 methods will be shown, all of which are fairly widely used in the United States.

(Note that the actual applications being compared ranged from about 800 function points in size to 1,300. The author has a proprietary method for mathematically adjusting application sizes to a fixed size in order to facilitate side-by-side comparisons.)

## Methodologies in Alphabetic Order

1. Agile with scrum
2. CMMI 1 with waterfall
3. CMMI 3 with iterative
4. CMMI 5 with spiral
5. Extreme programming (XP)
6. Object-oriented development
7. Pair programming with iterative
8. Proofs of correctness with waterfall
9. Rational unified process (RUP)
10. Team Software Process (TSP)

Since not every reader may be familiar with every method, here are short descriptions of the ones in the article:

## Agile with scrum:

The term "agile" is ambiguous and there are many flavors of agile. For this article the term is used for projects that more or less follow the 1997 agile manifesto, have embedded users to provide requirements, use user stories, divide projects into discrete sprints that are developed individually, and use the scrum concept and daily status meetings. Minimizing paperwork and accelerating development speeds are top goals of agile.

## CMMI 1 with waterfall:

The Capability Maturity Model Integrated™ (CMMI) of the Software Engineering Institute is a well-well known method for evaluating the sophistication of software development. CMMI 1 is the bottom initial level of the 5 CMMI levels and implies fairly chaotic development. The term "waterfall" refers to traditional software practices of sequential development starting with requirements and not doing the next step until the current step is finished.

## CMMI 3 with iterative:

The third level of the CMMI is called “defined” and refers to a reasonably smooth and well understood set of development steps. The term “iterative” is older than “agile” but has a similar meaning of dividing applications into separate pieces that can be constructed individually.

## CMMI 5 with spiral:

The 5th level of the CMMI is the top and is called “optimizing.” Groups that achieve this level are quite sophisticated and seldom make serious mistakes. The spiral model of software development was pioneered by Dr. Barry Boehm. It features ideas that also occur in agile, such as individual segments that can be developed separately. The spiral segments are often larger than agile segments, and are preceded by prototypes.

## Extreme Programming (XP):

This method falls under the umbrella of agile development but has some unique features. The most notable unique feature is to delay coding until test cases have been developed first. The XP method also uses reviews or inspections. Sometimes pair programming is used with XP but not always, so that is a special case. Quality of the final software is a major goal of the XP method.

## Object-oriented development (OO):

The OO method is one of the oldest in this article and has had many successes. It has also led to the creation of special languages such as Objective C. In this article OO analysis and design with use cases are used. The C++ language is also an OO language. OO analysis and design are somewhat different from conventional methods so a learning curve is needed.

## Pair programming:

The concept of pair programming is often part of the agile approach, but is not limited to it. In this paper pair programming is used with iterative development. The basic idea of pair programming is that two people take turns coding. While one is coding the other is watching and making suggestions. Sometimes the pair use only one computer or work station between them.

## Proofs of correctness:

The concept behind proofs of correctness is that of applying formal mathematical proofs to the algorithms that will be included in a software application. It is obvious that the algorithms need to be expressed in a formal manner so that they can be proved. It is also obvious that the person who performs the proof has enough mathematical skills to handle rather complex equations and algorithms.

## Rational Unified Process (RUP):

The RUP methodology was originated by the Rational Corporation which was acquired by IBM in 2003 so it is now an IBM methodology. The RUP method includes aspects of both iterative and object-oriented development. Since RUP is now owned by IBM there are numerous tools that support the method. Use Cases and visual representations are standard

for RUP applications, but the author's clients usually include other methods as well such as decision tables.

## Team Software Process (TSP):

The TSP method was developed by the late Watts Humphrey, who was IBM's director of programming and later created the assessment method used by the Software Engineering Institute (SEI) capability maturity model. TSP is very focused on software quality. All bugs are recorded; inspections are used, and high quality is the main goal on the grounds that bugs slow down development. The TSP method has some unusual aspects such as self-governing tools and a coach that serves the role of manager. TSP is now endorsed and supported by the SEI.

## Three Kinds of Methodology Evaluation and 10 Metrics

Even with the number of methods limited to 10 there are still a great many results that need to be evaluated. However from working with hundreds of clients, the topics that have the greatest importance to development managers and higher executives are these:

### *Speed-related metrics*

1. Development schedules
2. Development staffing
3. Development effort
4. Development costs

### *Quality-related metrics*

5. Defect potentials
6. Defect removal efficiency (DRE)
7. Delivered defects
8. High-severity defects

### *Economic-related metrics*

9. Total Cost of Ownership (TCO)
10. Cost of Quality (COQ)

Even with only 10 methodologies and 10 topics to display, that is still quite a significant amount of information.

This article will attempt to compare methodologies in three major categories:

1. Speed: Development schedules, effort, and costs

2. Quality: Software quality in terms of delivered defects

3. Economics: Total Cost of Ownership (TCO) and Cost of Quality (COQ)

Note that the technique used in this article of holding application size constant at 1000 function points means that the data cannot be safely used to determine the best methods for

large systems of 10,000 function points or massive systems of 100,000 function points. However applications in the 1000 function point size range are very common, and are large enough to show comparative results in a fairly useful way.

Some of the data in this article was prepared using the author's Software Risk Master ™ (SRM) tool, which is designed to perform side-by-side comparisons of any development methodology, any CMMI level, any complexity level, and any level of team experience. Some of the tables are based on SRM outputs, although derived from earlier measured applications.

---

## Speed: Comparing Methodologies for Development Schedules and Costs

The first comparison of methodologies concerns initial development speeds, costs, and short-term issues. Among the author's clients the most frequent request when estimating software projects is to predict the development schedule. Because schedules are viewed as critical to a majority of software managers and executives, table 1 is sorted by the speed of development.

Table 1: Software Schedules, Staff, Effort, Productivity

| | **Methodologies** | **Schedule** | **Staff** | **Effort** | **FP** | **Development** |
|---|---|---|---|---|---|---|
| | | Months | | Months | Month | Cost |
| 1 | Extreme (XP) | 11.78 | 7 | 84 | 11.89 | $630,860 |
| 2 | Agile/scrum | 11.82 | 7 | 84 | 11.85 | $633,043 |
| 3 | TSP | 12.02 | 7 | 86 | 11.64 | $644,070 |
| 4 | CMMI 5/ spiral | 12.45 | 7 | 83 | 12.05 | $622,257 |
| 5 | OO | 12.78 | 8 | 107 | 9.31 | $805,156 |
| 6 | RUP | 13.11 | 8 | 101 | 9.58 | $756,157 |
| 7 | Pair/iterative | 13.15 | 12 | 155 | 9.21 | $1,160,492 |
| 8 | CMMI 3/iterative | 13.34 | 8 | 107 | 9.37 | $800,113 |
| 9 | Proofs/waterfall | 13.71 | 12 | 161 | 6.21 | $1,207,500 |
| 10 | CMMI 1/waterfall | 15.85 | 10 | 158 | 6.51 | $1,188,870 |
| | Average | 13 | 8.6 | 112.6 | 9.762 | $844,852 |

As can be seen the software development methods that yield the shortest schedules for applications of 1000 function points are the XP and Agile methods, with TSP coming in third.

---

## Quality: Comparing Defect Potentials, Defect Removal, and Delivered Defects

The next topic of interest when comparing methodologies is that of quality. The article considers four aspects of software quality: defect potentials, defect removal efficiency, delivered defects, and high-severity defects.

The phrase "defect potential" refers to the sum of defects found in requirements, design, source code, documents, and "bad fixes." A bad fix is a new defect accidentally injected

during an attempt to repair a previous defect. (About 7% of attempts to fix bugs include new bugs.)

The phrase "defect removal efficiency" refers to the combined efficiency levels of inspections, static analysis, and testing. In this article six kinds of testing were included: 1) unit test; 2) function test; 3) regression test; 4) performance test; 5) system test; 6) acceptance test.

(There are about 40 total kinds of testing, but the specialized forms of testing are outside the scope of this article.)

When quality is evaluated readers can see why the parable of the blind man and the elephant was cited earlier:

### Table 2: Software Defect Potentials, Removal, and Delivery

| | Methodologies | Defect Potential | Defect Removal | Defects Delivered | Hi Sev. Defects |
|---|---|---|---|---|---|
| 1 | TSP | 2,700 | 96.79% | 87 | 16 |
| 2 | CMMI 5/ spiral | 3,000 | 95.95% | 122 | 22 |
| 3 | RUP | 3,900 | 95.07% | 192 | 36 |
| 4 | Extreme (XP) | 4,500 | 93.36% | 299 | 55 |
| 5 | OO | 4,950 | 93.74% | 310 | 57 |
| 6 | Pair/iterative | 4,700 | 92.93% | 332 | 61 |
| 7 | Proofs/waterfall | 4,650 | 92.21% | 362 | 67 |
| 8 | Agile/scrum | 4,800 | 92.30% | 370 | 68 |
| 9 | CMMI 3/ Iter. | 4,500 | 91.18% | 397 | 73 |
| 10 | CMMI 1/ Water. | 6,000 | 78.76% | 1,274 | 236 |
| | Average | 4,370 | 92.23% | 374 | 69 |

When the focus of the evaluation turns to quality rather than speed, TSP, CMMI 5, and RUP are on top, followed by XP. Agile is not strong on quality so it is only number 8 out of 10. The Agile lack of quality measures and failure to use inspections will also have an impact in the next comparison.

---

## Economics: Total Cost of Ownership (TCO) and Cost of Quality (COQ)

Some of the newer methods such as Agile and XP have not been in use long enough to show really long-range findings over 10 years or more. In this article TCO is limited to only five years of usage, because there is almost no data older than that for Agile.

The figures for TCO include development, five years of enhancement, five years of maintenance or defect repairs, and five years of customer support. While the Software Risk Master tool predicts those values separately, in this article they are all combined together into a single figure.

The figures for COQ consolidate all direct costs for finding and fixing bugs from the start of requirements through five years of customer usage.

## Table 3: Total Cost of Ownership (TCO): Cost of Quality (COQ)

| | Methodologies | TCO | COQ | Percent |
|---|---|---|---|---|
| 1 | TSP | $1,026,660 | $298,699 | 29.09% |
| 2 | CMMI 5/ spiral | $1,034,300 | $377,880 | 36.53% |
| 3 | Extreme (XP) | $1,318,539 | $627,106 | 47.56% |
| 4 | RUP | $1,360,857 | $506,199 | 37.20% |
| 5 | Agile/scrum | $1,467,957 | $774,142 | 52.74% |
| 6 | OO | $1,617,839 | $735,388 | 45.45% |
| 7 | CMMI 3/iterative | $1,748,043 | $925,929 | 52.97% |
| 8 | Pair/iterative | $2,107,861 | $756,467 | 35.89% |
| 9 | Proofs/waterfall | $2,216,167 | $863,929 | 38.98% |
| 10 | CMMI 1/waterfall | $3,944,159 | $2,804,224 | 71.10% |
| | Average | $1,784,238 | $866,996 | 44.75% |

Because applications developed using the TSP, CMMI 5, and RUP methodologies are deployed with low numbers of defects it is fairly easy to enhance them, maintain them, and support them. Therefore the 5-year total cost of ownership clearly favors the quality-related methods rather than the speed-related methods.

Agile is not bad, but with a COQ of more than 50% Agile needs to take quality more seriously up front.

The COQ percentages reveal a chronic problem for software applications. We have so many bugs that finding and fixing bugs is the major cost of both development and total cost of ownership.

---

## The Methods that Achieve Top Rankings in all Categories

To continue with the metaphor of the blind men and the elephant, here are the top methods in each of the 10 categories:

## Table 4: Top Methods in 10 Categories

| 1 | Development schedules | Extreme programming (XP) |
|---|---|---|
| 2 | Development staffing | Agile/scrum (tied) |
| 3 | Development effort | CMMI/5 spiral |
| 4 | Development costs | CMMI/5 spiral |
| 5 | Defect potentials | TSP |
| 6 | Defect removal efficiency (DRE) | TSP |
| 7 | Delivered defects | TSP |
| 8 | High-severity defects | TSP |
| 9 | Total Cost of Ownership (TCO) | TSP |
| 10 | Cost of Quality (COQ) | TSP |

The phrase “be careful of what you wish for because you might get it” seems to be appropriate for these methodology comparisons. Methods such as Agile that focus on speed are very quick. Methods such as TSP, RUP, and CMMI 5 that focus on quality have very few defects.

## Why some Methods Compare Poorly for Speed, Quality, and Economics

As can be seen the various methodologies fluctuated in their effectiveness on the speed, quality, and economic dimensions. However three methodologies were near the bottom for all three evaluations. These laggards were the waterfall method, which was in last place, the proof of correctness method, and the pair programming method. It is useful to explain the probable reasons for the low placements of these three methodologies.

## Waterfall and CMMI 1

It is no secret that about 35% of software projects for more than 50 years have been cancelled due to poor quality or overruns. Most of these used waterfall development and either were at CMMI level 1 or did not use the CMMI at all.

At the 1000 function point size range used in this example for waterfall, the percentage of time and effort devoted to finding and fixing bugs is about 25.71%. The number of projects that run late or exceed their budgets is about 50%. These are not very large applications, but with waterfall they are often troublesome.

It should be mentioned that the primary motivation of most of the newer methods is to overcome the historical problems associated with waterfall development.

There have been a few successes with waterfall projects but these tend to be those done by expert teams.

---

## Pair Programming

Unfortunately pair programming is an expensive mistake. The idea of letting two people take turns programming while one watched is a theoretical idea but weak in practice. The evidence for pair programming is flawed. There are assertions that pairs create software with fewer bugs than individual programmers. Unfortunately the individuals were using basic waterfall methods. Capable individual programmers who use static analysis and participate in formal code inspections of their work produce better code for about half the cost of pair programming.

Further, there are some 90 different software occupations. Why double up programmers and nobody else? If the idea of pair programming worked as asserted, then architects, business analysts, testers, quality assurance and everybody might be doubled. Why not use two project managers instead of one?

The usage of pair programming is symptomatic of very poor measurement practices and also a failure to understand the distribution of talent among large populations. If a company were building a large system with 500 programmers, it would not be possible to bring in or hire 500 more to pair up with them.

### Proofs of Correctness

The idea of proofs of correctness is an academic construct and is more theoretical than real. In order to prove algorithms in software they need to be formally expressed and the personnel doing the proofs need considerable mathematical sophistication. Even then, errors will occur in many proofs.

In the sample used in this article for 1000 function points there were about 690 specific requirements that need to be proved. This is why even small applications that use proofs take a long time, because proofs are time consuming.

It would be essentially impossible to use proofs of correctness on an application of 10,000 function points because there would be 7,407 specific algorithms to be proved and that might take several years, during which the requirements would have changed so many that the earlier proofs might no longer apply.

### Matching Software Methodologies with Projects

Since no method is top-ranked in every category, readers may well ask how to select methods that match the needs of their projects.

For smaller applications of 1000 function points or less where speed of delivery is the most critical parameter, then XP, Agile, and TSP are all very good choices.

For complex applications that might need FDA approval, operate weapons systems, or control sensitive financial data, high quality levels are mandatory. In this class TSP, CMMI 5, and RUP are the top choices, with XP as another possible method. Agile has been used for such applications but needs to be bent and twisted so much that it no longer is very agile. Agile is not strong on quality.

For applications that might last for more than 10 years or which require very frequent enhancements and therefore need well designed interior structures, TSP would be the top choice, with CMMI 5, RUP, and XP also possibilities. Agile has not shown much success with long-term maintenance and enhancements.

---

# SUMMARY AND CONCLUSIONS

As this article is written the software industry has about 55 different development methodologies. This is too large a number to compare in a short article.

For the 10 methods compared here, most have had some successes and most have had a few failures too.

Overall the Agile family and the methods that emphasize speed have achieved their goal, and they are fairly quick.

The methods that emphasize quality such as TSP, RUP, and CMMI 5 have also achieved their goals, and deliver very few defects.

No single method appears to be a universal panacea that can be successful on every size and kind of software application.

This article attempts to show the methods that give the best fit to three important factors: 1) speed; 2) quality; 3) long-rang economic value.

---

# REFERENCES AND READINGS


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).