# Development Practices for Small Software Applications©

Capers Jones
*Software Productivity Research, LLC*

*Because large software projects are troublesome and often out get out of control, a number of effective development methods have been created to help reduce the problems of large software projects. Two of these methods include the well-known Capability Maturity Model® (CMM®) and the newer CMM Integration℠ (CMMI®) of the Software Engineering Institute (SEI). When these methods are applied to small projects, it is usually necessary to customize them because, in their original form, they are somewhat cumbersome for small projects, although proven effective for large applications. More recently, alternate methods derived from smaller software projects have become popular under the general name of Agile software development. Since the Agile methods originated for fairly small projects, they can be applied without customization and often return very positive results. It is possible to merge Agile concepts with CMM and CMMI concepts, but projects cited in this article did not do so because such mergers are comparatively rare.*

Software applications come in a wide range of sizes and types. Each size and type tends to have evolved typical development practices. For example, military projects are usually much more formal and perform many more oversight and control activities than civilian projects. Systems software tends to have more kinds of testing and quality control activities than information systems.

When application sizes are considered, there are very significant differences in the development processes that are most commonly used for large software projects compared to small software projects.

Before discussing size differences, it is best to start by defining what is meant by the terms *large* and *small*. Although there is no agreed-to definition for these terms, the author generally regards *large* applications as being those whose size exceeds 10,000 function points or about 1,000,000 source code statements. The author regards *small* applications as those whose size is at or below 1,000 function points or about 100,000 source code statements.

It is a significant observation that out of 16 software lawsuits where the author served as an expert witness, 15 of them were for applications in the 10,000 function point size range, or larger. The smallest application noted during litigation was 3,000 function points.

Usually software applications of 1,000 function points or below are fairly trouble-free and, therefore, seldom end up in litigation for outright failure, quality problems, cost overruns, or schedule slips. On the other hand, an alarming percentage of applications larger than 10,000 function points exhibit serious quality problems, outright failure, or massive overruns.

Some readers might think that 1,000 function points or 100,000 source code statements are still rather large. However, there are very few commercial software applications or business applications that are smaller than this size range. Software in the range of 100 function points or 10,000 source code statements usually consists of enhancements to larger applications rather than stand-alone applications. Even the smallest commercial off-the-shelf software packages are in the range of 1,000 function points.

> *"When application sizes are considered, there are very significant differences in the development processes that are most commonly used for large software projects compared to small software projects."*

The overall size range of software applications noted by the author runs from a high of about 300,000 function points (30,000,000 source code statements) down to about 0.1 function points (10 source code statements) for a small bug repair.

At the extreme high end are very few massive applications such as enterprise resource planning (ERP) packages and some large defense systems. Operating systems and major application packages such as Microsoft Office are in the range of 100,000 function points or 10,000,000 source code statements. Various components of Microsoft Office such as Excel, Word, PowerPoint, etc., are between 5,000 and 10,000 function points in size.

## Origins of the CMM and Agile Development

The SEI was incorporated in 1984 and is located at Carnegie Mellon University in Pittsburgh, Pennsylvania. The SEI was originally funded by the Defense Advanced Research Projects Agency.

Some of the major concerns that led to the creation of the SEI were the very serious and common problems associated with large software projects. One of the early, major, and continuing activities of the SEI was the development of a formal evaluation or assessment schema for evaluating the capabilities of defense contractors, which was termed CMM. Watts Humphrey's book on this topic, "Managing the Software Process" became a bestseller [1].

The initial version of the CMM was published by SEI in 1987, and continued to grow and evolve for about 10 years. Development of the CMM more or less stopped in 1997 and the emphasis switched to the next generation, or the CMMI. The CMMI was initially published in 2002 and has been updated several times. Associated with the CMMI are Watts Humphrey's Team Software Process℠ (TSP℠) and Personal Software Process℠ (PSP℠).

The SEI assessment approach is now


® Capability Maturity Model, CMM and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.
℠ CMM Integration, Team Software Process, Personal Software Process, TSP, and PSP are service marks of Carnegie Mellon University.

well documented and well covered in software literature. Indeed, Addison Wesley Longman has an entire series devoted to SEI assessments. The SEI assessment data is collected by means of on-site interviews using both a standard questionnaire and also observations and informal data.

Because of the importance of very large systems to the Department of Defense, the SEI assessment approach originally dealt primarily with the software processes and methodologies used by large companies that produced large systems. The original SEI assessment approach was derived from the best practices used by leading corporations such as IBM and ITT, which employ from 5,000 to more than 50,000 software professionals, and which could safely build systems in excess of 1,000,000 lines of code or 10,000 function points.

Based on the patterns of answers to the SEI assessment questions, the final result of the SEI assessment process places the software organization on one of the levels of a five-point maturity scale. The five plateaus of the SEI maturity levels are shown in Table 1.

It is immediately obvious that the distribution of software organizations is skewed toward the low end of the scale. A similar kind of skew would occur if you were to look at the distribution of candidates selected to enter the Olympics for events such as downhill skiing. Most ordinary citizens could not qualify at all. Very few athletes could make it to the Olympic tryouts, even fewer would represent their countries, and only three athletes from around the world will win medals in each event.

As data is collected, it becomes evident that there is quite a bit of overlap among the various SEI maturity levels. For example, in terms of both quality and productivity, the best software projects from Level 1 organizations can be superior to the worst developed by Level 3 organizations, although the statistical averages of Level 3 are far better than those of Level 1.

There is now fairly solid evidence about the CMM from many studies. When organizations move from CMM Level 1 up to Level 2, 3, 4, and 5, their productivity and quality levels tend to improve based on samples at each level.

As of 2007, the newer CMMI has less empirical data than the older CMM, which is not surprising given its more recent publication date. However, the TSP and PSP methods do have enough data to show that they are successful in improving both quality and productivity at the same time.

When the CMM originated in the 1980s, the *waterfall* method of development was the most common at that time and was implicitly supported by most companies that used the early CMM. However, other methods such as spiral, iterative, etc., were quickly included in the CMM as well. The CMM is neutral as to development methods, but among the author's clients who adopted the CMM, about 80 percent also used the waterfall method.

*"What the CMM provided was a solid framework of activities, much better rigor in the areas of quality control and change management, and much better measurement of progress, quality, and productivity than was previously the norm."*

What the CMM provided was a solid framework of activities, much better rigor in the areas of quality control and change management, and much better measurement of progress, quality, and productivity than was previously the norm.

Table 1: *Five Levels of the CMM*

| SEI Maturity Level | Meaning | Frequency of Occurrence |
|---|---|---|
| 1 = Initial | Chaotic | 75.0% |
| 2 = Repeatable | Marginal | 15.0% |
| 3 = Defined | Adequate | 8.0% |
| 4 = Managed | Good to excellent | 1.5% |
| 5 = Optimizing | State of the art | 0.5% |

The history of the Agile methods is not as clear as the history of the CMM because the Agile methods are somewhat diverse. However, in 2001 the famous Agile Manifesto was published [2]. This provided the essential principles of Agile development. That being said, there are quite a few Agile variations including eXtreme Programming (XP), Crystal Development, Adaptive Software Development, Feature Driven Development, and several others.

Some of the principle beliefs found in the Agile Manifesto include the following:

- Working software is the goal, not documents.
- Working software is the principle measure of success.
- Close and daily contact between developers and clients is necessary.
- Face-to-face conversation is the best form of communication.
- Small, self-organizing teams give the best results.
- Quality is critical, so testing should be early and continuous.

The Agile methods, the CMM, and the CMMI are all equally concerned about three of the same fundamental problems:

1. Software requirements always change.
2. Fixing software bugs is the most expensive software activity in history.
3. High quality leads to high productivity and short schedules.

However, the Agile method and the CMM/CMMI approach draw apart on two other fundamental problems:

1. Paperwork is the second most expensive software activity in history.
2. Without careful measurements, continuous progress is unlikely.

The Agile method takes a strong stand that paper documents in the form of rigorous requirements and specifications are too slow and cumbersome to be effective. In the Agile view, daily meetings with clients are more effective than written specifications. In the Agile view, daily team meetings or Scrum sessions are the best way of tracking progress, as opposed to written status reports. The CMM and CMMI do not fully endorse this view.

The CMM and CMMI take a strong stand that measurements of quality, productivity, schedules, costs, etc., are a necessary adjunct to process improvement and should be done well. In the view of the CMM and CMMI, it is hard to prove that a methodology is a success or not without data that demonstrates effective

progress. The Agile method does not fully endorse this view. In fact, one of the notable gaps in the Agile approach is any quantitative quality or productivity data that can prove the success of the methods.

## Differences in Development Activities Between the Agile and CMM/CMMI Methods

In many industries, building a large product is not the same as building a small product. Consider the differences in specialization and methods required to build a wooden kayak versus building an 80,000-ton cruise ship. A kayak can be constructed by a single individual using only hand tools, but a large, modern cruise ship requires more than 500 workers including specialists such as pipe fitters, electricians, steel workers, welders, painters, interior decorators, air conditioning specialists, and many others.

Software follows a similar pattern: Building a large system in the 10,000 function point range is more or less equivalent to building other large structures such as ships, office buildings, or bridges. Many kinds of specialists are utilized and the development activities are quite extensive compared to smaller applications.

Because the CMM approach was developed in the 1980s when the waterfall method was common, it is not difficult to identify the major activities that are typically performed. For an application of 1,000 function points (approximately 100,000 source code statements), the following 20 normal activities were noted among the author's clients who used either the CMM or CMMI:

1. Requirements.
2. Prototyping.
3. Architecture.
4. Project planning and estimating.
5. Initial design.
6. Detailed design.
7. Design inspections.
8. Coding.
9. Reuse acquisition.
10. Code inspections.
11. Change and configuration control.
12. Software quality assurance.
13. Integration.
14. Test plans.
15. Unit testing.
16. New function testing.
17. Regression testing.
18. Integration testing.
19. Acceptance testing.
20. Project management.

Using the CMM and CMMI, the entire application of 1,000 function points would have the initial requirements gathered and analyzed, the specifications written, and various planning document produced before coding got under way.

By contrast, the Agile method of development would follow a different pattern. Because the Agile goal is to deliver running and usable software to clients as rapidly as possible, the Agile approach would not wait for the entire 1,000 function points to be designed before coding started.

What would be most likely with the Agile methods would be to divide the overall project into four smaller projects, each of about 250 function points in size. In Agile terminology, these smaller segments are termed iterations or sometimes *sprints*.

However, in order to know what the overall general set of features would be, an Agile project would start with *Iteration 0* or a general planning and requirements-gathering session. At this session, the users and developers would scope out the likely architecture of the application and then subdivide it into a number of iterations.

Also, at the end of the project when all of the iterations have been completed, it will be necessary to test the combined iterations at the same time. Therefore, a release phase follows the completion of the various iterations. For the release, some additional documentation may be needed. Also, cost data and quality data need to be consolidated for all of the iterations. A typical Agile development pattern might resemble the following:

- **Iteration 0**
  1. General overall requirements.
  2. Planning.
  3. Sizing and estimating.
  4. Funding.
- **Iterations 1-4**
  1. User requirements for each iteration.
  2. Test planning for each iteration.
  3. Testing case development for each iteration.
  4. Coding.
  5. Testing.
  6. Scrum sessions.
  7. Iteration documentation.
  8. Iteration cost accumulation.
  9. Iteration quality data.
- **Release**
  1. Integration of all iterations.
  2. Final testing of all iterations.

### Successful Hybrid Approaches to Software Development

One of the major trends in the industry since about 1997 has been to couple the most effective portions of various software development methodologies to create hybrid approaches. These hybrid methods are often successful and often achieve higher quality and productivity rates than the original *pure* methods. As of 2007 some interesting examples of the hybrid approach include, in alphabetical order:

- Agile joined with object-oriented development (OOD).
- Agile joined with Lean Six-Sigma.
- Agile joined with TSP and PSP.
- Agile joined with the CMM and CMMI.
- CMM and CMMI joined with Six-Sigma.
- CMM and CMMI joined with International Standardization for Organization (ISO) standard certification.
- CMM and CMMI joined with Information Technology Infrastructure Library (ITIL).
- CMM and CMMI joined with Quality Function Deployment (QFD).
- CMM and CMMI joined with TSP and PSP.
- XP joined with Lean Six-Sigma.
- ITIL joined with Six-Sigma.
- ISO joined with TickIT.
- OOD joined with service-oriented architecture (SOA).
- Six-Sigma joined with QFD.
- Six-Sigma joined with ITIL.
- Six-Sigma joined with TSP/PSP.
- SOA joined with ITIL.
- TickIT joined with Six-Sigma.

The list above only links pairs of development methods that are joined together. From time to time three or even four or five development practices have been joined together:

- Agile joined with OOD, joined with Lean Six-Sigma, and joined with ITIL.
- CMM joined with Agile, joined with TSP/PSP, joined with Six-Sigma, and joined with QFD.

| | Agile | CMM Level 3 | Difference |
|---|---|---|---|
| Size in function points | 1,000 | 1,000 | 0 |
| Size in Java code statements | 50,000 | 50,000 | 0 |
| Monthly burdened cost | $7,500 | $7,500 | 0 |
| Work hours per month | 132 | 132 | 0 |
| Project staff | 5 | 7 | 2 |
| Project effort (months) | 66 | 115 | 49 |
| Project effort (hours) | 8,712 | 15,180 | 6,486 |
| Project schedule (months) | 14 | 19 | 5 |
| Project cost | $495,000 | $862,500 | $367,500 |
| Function points per month | 15.15 | 8.67 | -6.46 |
| Work hours per function point | 8.71 | 15.18 | 6.47 |
| LOC per month | 758 | 435 | -323 |
| Function point assignment scope | 200 | 143 | -57 |
| LOC assignment scope | 10,000 | 7,143 | -2,857 |
| Cost per function point | $495 | $863 | $368 |
| Cost per LOC | $9.90 | $17.25 | $7.35 |
| Defect potential | 4,250 | 4,500 | 250 |
| Defect potential per function point | 4.25 | 4.50 | 0.25 |
| Defect removal efficiency | 90% | 95% | 0.5% |
| Delivered defects | 425 | 225 | -200 |
| High-severity defects | 128 | 68 | -60 |
| Delivered defects per function point | 0.43 | 0.23 | 0.20 |
| Delivered defects per thousand LOC | 8.50 | 4.50 | -4.00 |

Table 2: *Comparison of Agile and CMM Results for an Application of 1,000 Function Points*

3. Acceptance testing of application.
4. Total cost accumulation.
5. Quality data accumulation.
6. Final Scrum session.

These are the most interesting and unique features of the Agile methods: 1) The decomposition of the application into separate iterations, 2) The daily face-to-face contact with one or more user representatives, 3) The daily *Scrum* sessions to discuss the backlog of work left to be accomplished and any problems that might slow down progress. Another interesting feature is to create the test cases before the code itself is written, which is a feature of XP.

## Comparative Results of Agile and CMM for an Application of 1,000 Function Points

There is much more quantitative data available on the results of projects using the CMM than for projects using the Agile methods. In part this is because the CMM and CMMI include measurement as a key practice. Also, while Agile projects do estimate and measure, some of the Agile metrics are unique and have no benchmarks available as of 2007. For example some Agile projects use *story points*, some use *running tested features*, some use *ideal time*, and some measure with *use case points*. There are no large collections of data using any of these metrics nor are there reliable conversion rules to standard metrics such as function points.

For this article, an artificial test bed will be used to examine the comparative results of Agile and the CMM. The test bed is based on a real application (an online survey tool), but the size has been arbitrarily set to exactly 1,000 function points or 50,000 Java statements.

A sample of 20 CMM projects examined by the author and his colleagues was condensed into an overall aggregate for the CMM results. For the Agile data, observations and discussions with practitioners on five projects were used to construct the aggregate results. This is not a particularly accurate and reliable way to perform comparative analysis. As it happens, none of the 20 CMM applications used Agile methods, nor did any of the Agile projects use aspects of the CMM or CMMI.

Merging aspects of the Agile and CMM concepts is not difficult. Indeed, some CMM and CMMI applications circa 2007 do use Agile features such as Scrum sessions. However, among the author's clients, about 85 percent of CMM/CMMI users do not use Agile and about 95 percent of Agile users do not make use of either the CMM or CMMI. For example, at a recent conference of more than 100 state software managers, more than 50 percent of current projects were using Agile methods – but no projects at all were using either the CMM or CMMI.

Though briefly discussed in the sidebar, the topic of hybrid approaches is not well covered in the software engineering literature. Neither is the topic of hybrid approaches well covered in terms of benchmarks and quantitative data, although some hybrid projects are present in the International Software Benchmark Standards Group data. To simplify the results in this article, the Agile methods were used in *pure* form as were the CMM and CMMI methods. Since the *pure* forms still outnumber hybrid approaches by almost 10-to-1, that seems like a reasonable way to present the data.

The sizes of the samples were not, of course, exactly 1,000 function points. They ranged from about 900 to almost 2,000 function points. Not all of the samples used Java either. Therefore the following comparison in Table 2 has a significant but unknown margin of error.

What the comparison does provide is side-by-side data points for a standard test bed, with the measurements for both sides expressed in the same units of measure. Hopefully future researchers from both the Agile and CMM/CMMI camps will be motivated to correct the results shown here, using better methods and data.

Note that the *lines of code* (LOC) metric is not reliable for economic studies since it penalizes high-level programming languages and cannot be used for cross-language comparisons. However, since many people still use this metric, it is provided here with the caveat that none of the LOC values would be the same for other programming languages such as C++, Smalltalk, Visual Basic, etc.

The data in Table 2 is expressed in terms of function point metrics and assumes version 4.2 of the counting rules published by the International Function Point Users Group.

The CMM version of the project is assumed to be developed by an organization at Level 3 on the CMM. The Agile version of the project is assumed to be developed by an experienced Agile team. However, the Agile version is assumed not to use either the CMM or CMMI and so has no level assigned. This is a reasonable assumption because very few Agile teams are also CMM certified.

Although one comparison is probably insufficient, observations of many

small projects in the 1,000 function point size range indicate that the CMM and CMMI are somewhat more cumbersome than the Agile methods and therefore tend to have somewhat lower productivity rates (see Table 2). That is not to say that the CMM/CMMI methods produce bad or unacceptable results, but only that the Agile methods appear to generate higher productivity levels.

However, if the size plateau for comparison is upped to 10,000 function points or 100,000 function points, the CMM/CMMI methods would pull ahead. At 10,000 function points the overall staff would be larger than 50, while for 100,000 function points the overall staff would approach 600 people and some of the Agile principles such as daily team meetings would become difficult and perhaps impossible.

Also, applications in the 10,000 to 100,000 function point size range tend to have thousands or even millions of users. Therefore it is no longer possible to have direct daily contact with users since their numbers are too large.

When quality is considered, the Agile approach of having frequent daily contacts with users, daily Scrum sessions, and writing the test cases before the code has the effect of lowering *defect potentials*. The phrase defect potentials refers to the total number of defects that might be encountered in requirements, design, coding, user documents, as well as *bad fixes* or secondary defects. (The current U.S. average would be about 5.0 defects per function point for defect potentials.) However, Agile projects usually do not perform formal design and code inspections so their defect removal efficiency is somewhat below the CMM example.

The defect potentials for the CMM version are better than U.S. averages but not quite equal to the Agile version due to the more detached connections between clients and developers.

However, the CMM and CMMI methods typically do utilize formal design and code inspections. As a rule of thumb, formal inspections are more than 65 percent efficient in finding bugs or defects, which is about twice the efficiency of most forms of testing, many of which only find about 30 percent of the bugs that are actually present.

As of 2007 the current U.S. average for cumulative defect removal efficiency is only about 85 percent, so both the Agile and CMM examples are better than U.S. norms. The CMM method is somewhat higher than the Agile method due to formal inspections, testing specialists, and a more formal quality assurance approach.

As a general rule, methods developed to support large software applications such as the CMM and CMMI are cumbersome when applied to small projects. They can be tailored or subset to fit small projects, but out of the box they are cumbersome.

On the other hand, methods developed to support small software applications such as the Agile methods become less and less effective as application sizes increase. Here too, they can be tailored or modified to fit larger projects, but out of the box they are not effective.

As of 2007 both the Agile and CMM/CMMI communities are working on merging the more useful features of both sets of methods. Agile and the CMM/CMMI are not intrinsically opposed to one another, they merely started at opposite ends of the size spectrum.

## Overall Observations on Software Development

With more than 13,000 projects examined, it can be stated categorically that there is no single method of development that is universally deployed or even universally useful. All software projects need some form of gathering requirements, some form of design, some kind of development or coding, and some forms of defect removal such as reviews, inspections, and testing.

In the author's studies more than 40 methods for gathering requirements, more than 50 variations in handling software design, more than 700 programming languages, and more than 30 forms of testing were noted among the projects examined. Scores of hybrid methods have also been noted.

Both the Agile methods and the CMM/CMMI methods have added value to the software industry. But care is needed to be sure that the methods selected are in fact tailored to the size range of the applications being constructed.◆

## References


1. Humphrey, Watts S. Managing the Software Process. Reading, MA: Addison-Wesley Longman, 1989.
2. Cockburn, Alistair. Agile Software Development. Boston, MA: Addison Wesley, 2001.


## Additional Reading


1. Caputo, Kim. CMM Implementation Guide. Boston, MA: Addison Wesley, 1998.
2. Garmus, David and David Herron. Measuring the Software Process: A Practical Guide to Functional Measurement. Englewood Cliffs, NJ: Prentice Hall, 1995.
3. Jones, Capers. "Defense Software Development in Evolution." CROSSTALK Nov. 2002.
4. Jones, Capers. Software Assessments, Benchmarks, and Best Practices. Boston, MA: Addison-Wesley Longman, 2000.
5. Jones, Capers. Estimating Software Costs. New York. McGraw Hill, 2007.
6. Jones, Capers. Conflict and Litigation Between Software Clients and Developers. Burlington, MA: Software Productivity Research (SPR), Inc., 2007.
7. Kan, Stephen H. Metrics and Models in Software Quality Engineering. 2nd ed. Boston, MA: Addison-Wesley Longman 2003.


## About the Author


**Capers Jones** is currently the chairman of Capers Jones and Associates, LLC. He is also the founder and former chairman of SPR, where he holds the title of Chief Scientist Emeritus. He is a well-known author and international public speaker, and has authored the books "Patterns of Software Systems Failure and Success," "Applied Software Measurement," "Software Quality: Analysis and Guidelines for Success," "Software Cost Estimation," and "Software Assessments, Benchmarks, and Best Practices." Jones and his colleagues from SPR have collected historical data from more than 600 corporations and more than 30 government organizations. This historical data is a key resource for judging the effectiveness of software process improvement methods. The total volume of projects studied now exceeds 12,000.

**Software Productivity Research, LLC**
**Phone: (877) 570-5459**
**Fax: (781) 273-5176**
**E-mail: capers.jones@spr.com, info@spr.com**