Variations in Software Development Practices

Capers Jones, *Software Productivity Research*

My colleagues and I at Software Productivity Research gathered data on approximately 12,000 software projects between 1984 and 2003. We did this primarily to perform assessments and benchmark studies on software development quality and productivity. Part of the data includes descriptions of each project's software development practices. We also collected information on software development activities, tools, programming languages, and specialized personnel on projects.

To more easily analyze this large volume of information, we separated software projects into distinct categories that share a nucleus of common development methods and personnel. Size and type of software applications are the two major categories we used.[1]

In our analysis, we used a list of 25 standard software development activities, checking off which of these activities were included in the projects we examined. Of course, such a general checklist isn't perfect. We often encounter activities that are either not on the list or performed infrequently. We also recorded which occupation groups are associated with software development. I use a shortened list of occupations in this article—these 30 are the most commonly noted software occupations in the US.

Research on over 18 years of software projects reveals that developing large systems involves substantially more activities and a greater variety of specialized personnel than developing smaller systems.

Categorizing software applications

Although size is somewhat ambiguous in the literature, we found that function point metrics[2–4] rather than lines-of-code metrics provide a useful framework. To analyze software development practices, we divide projects into six size plateaus, each an order of magnitude apart: 1, 10, 100, 1,000, 10,000, and 100,000 function points.

Software projects measuring less than 100 function points are usually enhancements to older, larger applications. Major new software

0740-7459/03/$17.00 © 2003 IEEE

applications for business purposes most often measure between 10,000 and 100,000 function points. For example, both the Microsoft Windows and IBM MVS operating systems are roughly 100,000 function points. Many business and technical software applications such as accounting systems and telephone switching systems are between 10,000 and 100,000 function points.

We define type by placing applications into families that share common constraints and development methods. For convenience, we identified six major software types for our studies:

- *Military*. These applications are built according to military or US Department of Defense standards. Although weapons systems come to mind as a primary form of military software, the category is actually much broader. For example, logistics and support software applications—and even those for payroll and accounting—use military standards.
- *Systems*. These applications control hardware devices such as computers, aircraft, telephone switches, and other physical devices and products. This type also includes software that's embedded in hardware devices, such as software that controls car fuel-injection systems or some manufacturing robots.
- *Commercial*. Developers build these applications for lease or sale to external customers. This category includes many personal computer applications such as word processing and spreadsheet programs. It also includes larger mainframe applications such as enterprise resource-planning packages.
- *Outsourced*. This category includes applications that one or several companies build for a specific client company under a contract. Because of contractual obligations and the possibility of litigation, outsourced projects have some additional activities in comparison to in-house development. For example, outsourced projects might include special requirements for handling scope changes or new requirements, special accounting and tracking requirements, and special protocols for dealing with errors or bugs.
- *Management information systems (MIS)*. Companies and government agencies build these applications to control major business functions such as accounting, marketing, sales, and personnel. This category includes many traditional mainframe applications such as accounting systems, order-entry systems, payroll applications, and sales-support applications. It also includes the more recent client-server and Web-based applications.
- *End-user development*. This category refers to small applications that various kinds of knowledge workers—such as accountants, engineers, or project managers—build for personal use. I won't discuss this application type in depth because the developers aren't full-time software personnel.

Variations and application size

In many industries, building large products isn't the same as building small products. Consider the differences in specialization and methods that building a rowboat versus building an 80,000-ton cruise ship requires. One person can construct a rowboat using only hand tools. But a large, modern cruise ship requires over 250 workers, including specialists such as pipe fitters, electricians, steel workers, painters, and even interior decorators.

Software follows a similar pattern: Building a large system in the 10,000–100,000 function point range is basically equivalent to building large structures such as ships, office buildings, and bridges. Such a project uses many kinds of specialists for extensive development activities.

We compare the development activities included in software projects from the six different size plateaus in Table 1.

Projects below the 1,000-function point plateau (roughly equivalent to 100,000 source code statements in a procedural language such as Cobol include less than half of the activities. However, large systems in the 10,000–100,000 function point range perform more than 20 of these activities.

When we examine the occupations associated with software projects, we find that large systems use many more kinds of specialists than small projects. We list the occupation groups noted by project size in Table 2.

From Tables 1 and 2, we can conclude that size strongly influences development activities and the need for specialized personnel. Generalists usually develop small projects using informal development methods. It usually takes

Generalists usually develop small projects using informal development methods.

Table 1

Development activities for six project size plateaus and types

	No. of function points						Project types					
Activity	**1**	**10**	**100**	**1,000**	**10,000**	**100,000**	**End user**	**MIS**	**Outsourced**	**Commercial**	**Systems**	**Military**
Requirements	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓
Prototyping				✓	✓	✓	✓	✓	✓	✓	✓	✓
Architecture					✓	✓			✓	✓	✓	✓
Project plans				✓	✓	✓		✓	✓	✓	✓	✓
Initial design		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓
Detail design			✓	✓	✓	✓		✓	✓	✓	✓	✓
Design reviews					✓	✓			✓	✓	✓	✓
Coding	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Reuse acquisition	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Package purchase					✓	✓		✓	✓	✓	✓	✓
Code inspections				✓	✓	✓			✓	✓	✓	✓
Independent verification & validation												✓
Change control				✓	✓	✓		✓	✓	✓	✓	✓
Formal integration				✓	✓	✓		✓	✓	✓	✓	✓
User documentation			✓	✓	✓	✓		✓	✓	✓	✓	✓
Unit testing	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Function testing			✓	✓	✓	✓		✓	✓	✓	✓	✓
Integration testing				✓	✓	✓		✓	✓	✓	✓	✓
System testing				✓	✓	✓		✓	✓	✓	✓	✓
Beta testing					✓	✓				✓	✓	✓
Acceptance testing				✓	✓	✓		✓	✓	✓	✓	✓
Independent testing												✓
Quality assurance						✓			✓	✓	✓	✓
Installation & training				✓	✓	✓		✓	✓	✓	✓	✓
Project management			✓	✓	✓	✓		✓	✓	✓	✓	✓
Total no. of activities	4	5	9	18	22	23	4	18	22	23	23	25

teams of specialized personnel using much more complicated methodologies to develop large systems.

Variations and types of software

The second factor that influences software development practices is the type of software being constructed. For example, the methods for building military software are very different from civilian norms.[5] The systems and commercial software domains also have fairly complex development activities compared to MIS activities, such as formal specification methods and highly formal quality-assurance and defect-tracking protocols. The outsource domain, due to contractual requirements, also uses a fairly extensive set of development activities.

Table 1 notes the differences in development activities across the six types of software we studied. The activities for outsourced, commercial, systems, and military software are somewhat more numerous than for MIS projects where development processes are often somewhat rudimentary.

Software type also significantly impacts the personnel and kinds of specialists involved in systems development. Overall, the systems and military domains tend to employ the most specialists, followed closely by the commercial domain. The MIS domain is more likely to use generalists who perform analysis, design, coding, and testing tasks.

Table 2 shows the kinds of occupation groups that we have noted across the six project types. It shows that commercial, systems, and military software require more specialized occupation groups than MIS and outsourced projects.

It's interesting that companies building systems and military software are more than twice as likely to have formal QA departments and testing specialists than companies building MIS software, as we've seen in our studies. We might attribute this to the fact that companies building complex physical devices have used QA departments since before computers even existed. When computers and software became part of the product, they naturally applied historical QA roles to software as well.

Table 2

Occupation groups for six project size plateaus and types

	No. of function points						Project types					
Occupation	**1**	**10**	**100**	**1,000**	**10,000**	**100,000**	**End user**	**MIS**	**Outsourced**	**Commercial**	**Systems**	**Military**
Architects					✓	✓			✓	✓	✓	✓
Configuration control			✓	✓	✓	✓		✓	✓	✓	✓	✓
Cost analysis					✓	✓				✓	✓	✓
Cost estimating					✓	✓			✓	✓	✓	✓
Customer support					✓	✓			✓	✓	✓	✓
Data administration					✓	✓		✓	✓	✓		✓
Data base design					✓	✓		✓	✓	✓		✓
Data quality					✓	✓						✓
Education & training					✓	✓			✓	✓	✓	✓
Function point analysis						✓		✓	✓			
Globalization										✓	✓	✓
Graphics					✓	✓				✓	✓	✓
Human factors						✓				✓	✓	✓
Integration				✓	✓	✓		✓		✓	✓	✓
Librarians			✓	✓	✓	✓		✓	✓	✓	✓	✓
Metrics & measures						✓		✓	✓		✓	✓
Networks					✓	✓				✓	✓	✓
Performance analysis						✓					✓	✓
Programming	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Project management				✓	✓	✓		✓	✓	✓	✓	✓
Project planning					✓	✓			✓	✓	✓	✓
Quality assurance				✓	✓	✓			✓	✓	✓	✓
Reliability						✓					✓	✓
Reusability						✓			✓		✓	✓
Security						✓				✓	✓	✓
Software engineering						✓					✓	✓
Standards					✓	✓					✓	✓
Systems analysis					✓	✓		✓	✓	✓	✓	✓
Technical writers				✓	✓	✓			✓	✓	✓	✓
Testing					✓	✓		✓	✓	✓	✓	✓
Total no. of occupation groups	1	1	3	7	21	29	1	11	18	22	26	29

Overall software development observations

After examining more than 12,000 projects, we can categorically state that no single development method is universally deployed. All software projects need forms of requirements gathering, design, development or coding, and defect removal, such as reviews, inspections, and testing. But the methods for handling these generic activities aren't uniform. In the projects we examined, we noted over 40 methods for gathering requirements, over 50 variations in handling software design, over 700 programming languages, and over 30 forms of testing.

We find some of the most interesting similarities associated with software classes. Apparently because of tradition, people building software projects within the same class tend to build them similarly. Here we present the more interesting class-related findings.

Military

Before 1994, military software development was unique and specialized due to stringent military standards that controlled almost every aspect of software design and development. In 1994, the US Department of Defense began the move to civilian best practices, which is still ongoing even in 2003. The US Air Force software journal *Crosstalk*, published by the Software Technology Support Center, is a good source of information on the evolution of military and defense software practices.

The military standards seemed to be based on a lack of trust between vendors and the military services. As a result, military projects had elaborate oversight criteria and extensive tracking and documentation requirements. For example, the volume of military requirements, specifications, and planning documents was two to three times larger than civilian

The commercial software world does have one fairly unique quality approach: external testing by dozens or even hundreds of customers.

projects of the same size. More than half the cost of building large defense projects went to the production of English words. About 400 words were written for every line of Ada code in typical military applications.

The military community has been fairly successful in building large and complex applications such as weapons systems. However, productivity rates are far lower than civilian norms. In recent years, the military domain started using the Capability Maturity Model created by Watts Humphrey and his colleagues at the Software Engineering Institute.[6] Many more military than civilian projects use the CMM, which is why military projects use both specialists and extensive sets of development activities.

Military projects are roughly equal to civilian systems software in defect removal efficiency—often with a 95 percent defect removal rate or better. However, military projects use two additional defect removal steps that civilian projects seldom use: independent verification and validation and independent testing. Some defense projects have used as many as 16 different kinds of testing—more than any civilian project we examined.

Systems

Because systems software controls complicated and expensive physical devices such as computers, aircraft, and telephone systems, quality is a key factor. The systems and embedded software projects we looked at were more likely to use formal QA departments and testing departments staffed by full-time test personnel. The systems software community is also most likely to use formal design and code inspections and deploy thorough quality measurement systems.

The systems software community has the highest defect removal efficiency levels. It's the only type to average over 95 percent in finding and removing bugs or defects prior to release. This is due in part to the use of reviews, inspections, and six to 12 different kinds of testing activities on major projects.

Commercial

The commercial software world, headed by Microsoft and including other vendors such as SAP, IBM, and Computer Associates, is fairly strong in change control, customer support, documentation, training, and globalization.

However, commercial software lags behind systems software in quality-control methodologies. The commercial world isn't as likely to use pretest design reviews and code inspections. Neither is it as likely to use formal QA teams or have rigorous quality measurement systems during development. On the other hand, measurements of customer-reported complaints and defects after release are fairly good. We're not saying that all commercial vendors have poor quality control. IBM, for example, has long led the world in quality control; it was the first company known to measure defect removal efficiency and the first to top 95 percent. But we examined software from more than 25 vendors, and the overall quality results aren't as good as they should be, only averaging around 90 percent in defect removal efficiency.

The commercial software world does have one fairly unique quality approach: external testing by dozens or even hundreds of customers. Known as *beta testing*, the commercial world has been using this method since the 1960s. (Systems software and some military projects also use beta testing.) Overall, defect removal efficiency in the commercial world is lower than the systems and military communities.

Outsourced

The outsource community includes major companies such as Electronic Data Systems, Computer Sciences Corporation, Accenture, IBM, Lockheed, and more. This community is a bit more sophisticated than the MIS community, which constitutes the main client base for outsourcers. The outsource community is likely to use formal planning and change control methods. Outsourcers also widely deploy requirements gathering and analysis using joint application design (JAD). This isn't surprising because requirements changes usually lead to cost increases and design and code changes. The outsource community is fairly good in quality control and quality measurements.

Outsourcers aren't as good as the systems community in quality but better than the MIS community. This is because outsourcers often use design reviews, code inspections, and testing specialists. Defect removal efficiency in the outsource world runs 90 to 94 percent.

Management information systems

By looking at the results of in-house development projects, it's easy to see why so many companies—such as banks and insurance

companies—consider outsourcing. The MIS community is very backwards in quality control. It's less likely than most to use formal design reviews and code inspections, formal QA groups, or testing specialists. Quality in the MIS world depends on fairly rudimentary testing, which the developers carry out themselves. MIS projects averaged about 85 percent defect removal efficiency for many years. Unfortunately, client-server projects and Web-based applications are often even lower, sometimes dipping below 80 percent.

The good news for the MIS community is that requirements gathering and analysis using JAD is both common and fairly successful. Even so, the measured rate at which requirements change can top 2 percent per calendar month during development.

Interestingly, this community is better than average in productivity measurements. It pioneered the use of function point metrics for productivity studies and is more likely than any other group to employ certified function-point counting personnel. The MIS domain has more productivity data using function-points collected and published than any other.

End-user development

There is little to be said about end-user development. Various knowledge workers create these small applications (all are less than 100 function points) for private use. These developments have no formal requirements and usually no design documentation. Amateur programmers write many end-user applications in languages such as Basic or Visual Basic—their traditional languages. They might create some applications using spreadsheets if they will serve financial purposes.

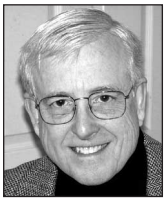
The originator performs any end-user testing. Although these projects seldom measure defect removal, when they do, it's usually less than 60 percent. Because the developer is also the user, these programs don't have user manuals. So once the originator changes jobs or retires, the applications are usually discarded. End-user applications are interesting to create but have limited or even negative business value.

Some of the observations and conclusions from our studies support commonsense notions. For example, we noted that software specialists appear to be valuable and raise the odds of software projects succeeding. We also noted that for projects larger than 1,000 function points, those with QA teams and formal defect-tracking methods have better chances for meeting planned schedules and planned budgets than projects with informal quality control.

A few observations were surprising and perhaps counterintuitive. For example, we didn't find that any specific design method or programming language guaranteed either a successful or troubled project outcome. However, projects that used almost any kind of formal design method tended to have better quality than similar projects with informal or amorphous design.

Perhaps the most significant observation is that good quality control is the best overall indicator of a successful project. Schedule delays and cost overruns most often occur when you discover during testing that the application has so many bugs that it doesn't work. Projects using QA teams, formal design and code inspections, and pretest defect tracking always had shorter testing cycles and therefore were more likely to be deliverable on schedule.

About the Author



Capers Jones is chief scientist emeritus of Software Productivity Research, a subsidiary of Artemis Management Solutions. His research interests include software cost and schedule estimating, software quality control methods, and software metrics and measurements. He received his BA in English from the University of Florida. He is a member of the IEEE Computer Society and a lifetime member of the International Function Point Users Group. Contact him at Software Productivity Research, 6 Lincoln Knoll Dr., Burlington, MA 01803; cjones@spr.com.

References

1. C. Jones, *Software Assessments, Benchmarks, and Best Practices*, Addison-Wesley, 2000.
2. International Function Point Users Group Web site, www.IFPUG.org.
3. S. Kan, *Metrics and Models in Software Quality Engineering*, 2nd Ed., Addison-Wesley, 2002.
4. D. Garmus and D. Herron, *Measuring the Software Process: A Practical Guide to Functional Measurements*, Prentice Hall, 1995.
5. C. Jones, "Defense Software in Evolution," *Crosstalk*, vol. 15, no. 11, Nov. 2002, pp. 26–29.
6. W.S. Humphrey, *Managing the Software Process*, Addison-Wesley, 1989.

For more information on this or any other computing topic, please visit our Digital Library at http://computer.org/publications/dlib.