Software Cost Estimating Methods for Large Projects©

Capers Jones, Software Productivity Research, LLC
http://www.stsc.hill.af.mil/CrossTalk/2005/04/0504Jones.html

For large projects, automated estimates are more successful than manual estimates in terms of accuracy and usefulness. In descending order, the costs of large projects include defect removal, production of paper documents, coding, project management, and dealing with new requirements that appear during the development cycle. In addition, successful estimates for large projects must be adjusted to match specific development processes, to match the experience of the development team, and to match the results of the programming languages and tool sets that are to be utilized. Simple manual estimates cannot encompass all of the adjustments associated with large projects.

Software has achieved a bad reputation as a troubling technology. Large software projects have tended to have a very high frequency of schedule and cost overruns, quality problems, and outright cancellations. While this bad reputation is often deserved, it is important to note that some large software projects are finished on time, stay within their budgets, and operate successfully when deployed.

The successful software projects differ in many respects from the failures and disasters [1]. One important difference is how the successful projects arrived at their schedule, cost, resource, and quality estimates in the first place. From an analysis of the results of using estimating tools published in "Estimating Software Costs" [2], using automated estimating tools leads to more accurate estimates. Conversely, casual or manual methods of arriving at initial estimates are usually inaccurate and often excessively optimistic.

A comparison of 50 manual estimates with 50 automated estimates for projects in the 5,000-function point range showed interesting results [2]. The manual estimates were created by project managers who used calculators and spreadsheets. The automated estimates were also created by project managers or their staff-estimating assistants using several different commercial-estimating tools. The comparisons were made between the original estimates submitted to clients and corporate executives, and the final accrued results when the applications were deployed.

Only four of the manual estimates were within 10 percent of actual results. Some 17 estimates were optimistic by between 10 percent and 30 percent. A dismaying 29 projects were optimistic by more than 30 percent. That is to say, manual estimates yielded lower costs and shorter schedules than actually occurred, sometimes by significant amounts. (Of course several revised estimates were created along the way. But the comparison was between the initial estimate and the final results.)

In contrast, 22 of the estimates generated by commercial software estimating tools were within 10 percent of actual results. Some 24 were conservative by between 10 percent and 25 percent. Three were conservative by more than 25 percent. Only one automated estimate was optimistic, by about 15 percent.

One of the problems with performing studies such as this is the fact that many large projects with inaccurate estimates are cancelled without completion. Thus, for projects to be included at all, they had to be finished. This criterion eliminated many projects that used both manual and automated estimation.

Interestingly, the manual estimates and the automated estimates were fairly close in terms of predicting coding or programming effort. But the manual estimates were very optimistic when predicting requirements growth, design effort, documentation effort, management effort, testing effort, and repair and rework effort. The conclusion of the comparison was that both manual and automated estimates were equivalent for actual programming, but the automated estimates were better for predicting non-coding activities.

This is an important issue for estimating large software applications. For software projects below about 1,000 function points in size (equivalent to 125,000 C statements), programming is the major cost driver, so estimating accuracy for coding is a key element. But for projects above 10,000 function points in size (equivalent to 1,250,000 C statements) both defect removal and production of paper documents are more expensive than the code itself. Thus, accuracy in estimating these topics is a key factor.

Software cost and schedule estimates should be accurate, of course. But if they do differ from actual results, it is safer to be slightly conservative than it is to be optimistic. One of the major complaints about software projects is their distressing tendency to overrun costs and planned schedules. Unfortunately, both clients and top executives tend to exert considerable pressures on managers and estimating personnel in the direction of optimistic estimates. Therefore, a hidden corollary of successful estimation is that the estimates must be defensible. The best defense is a good collection of historical data from similar projects.

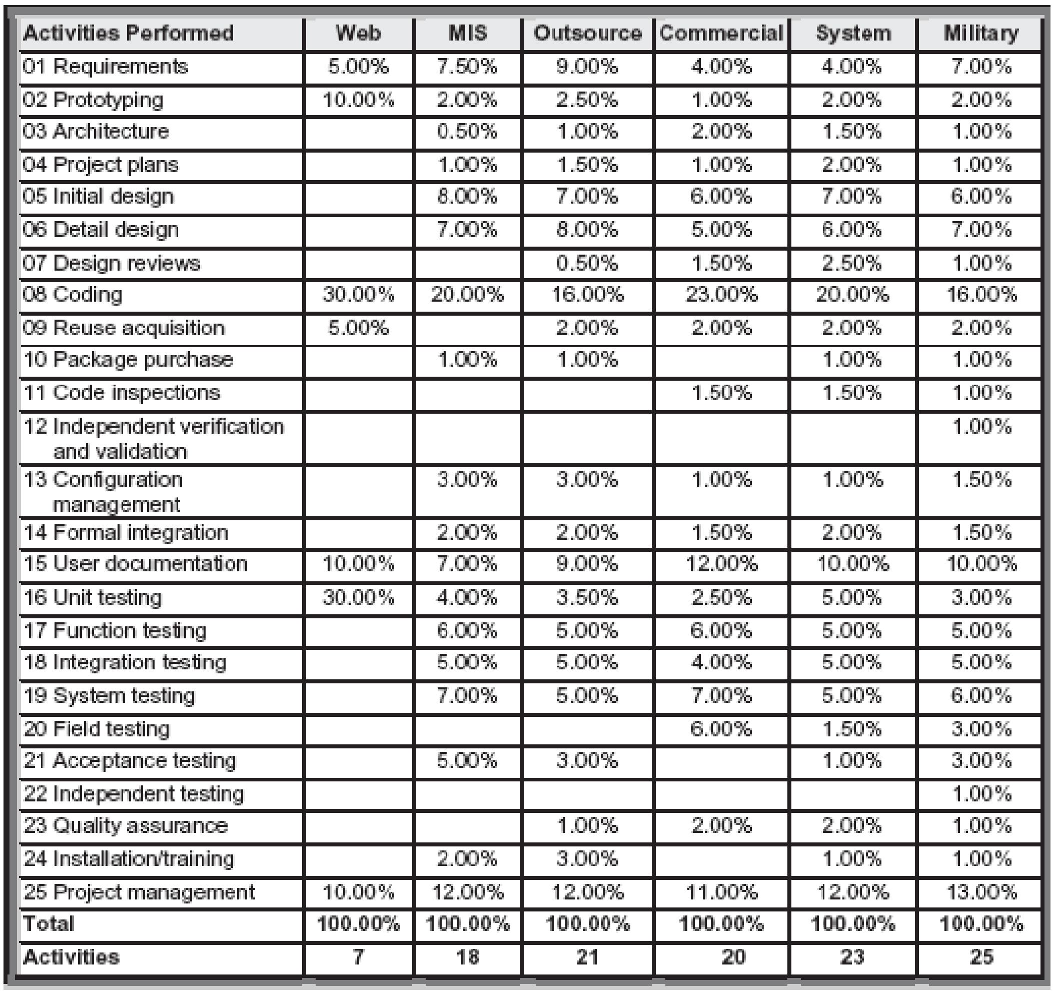
Because software estimation is a complex activity there is a growing industry of companies that market commercial software estimation tools. As of 2005, some of these estimating tools include COCOMO II, CoStar, CostModeler, CostXpert, KnowledgePlan, PRICE S, SEER, SLIM, and SoftCost. Some older automated costestimating tools are no longer being actively marketed but are still in use such as CheckPoint, COCOMO, ESTIMACS, REVIC, and SPQR/20. Since these tools are not supported by vendors, usage is in decline.

While these estimating tools were developed by different companies and are not identical, they do tend to provide a nucleus of common functions. The major features of commercial software-estimation tools circa 2005 include these attributes:

- Sizing logic for specifications, source code, and test cases.
- Phase-level, activity-level, and tasklevel estimation.
- Adjustments for specific work periods, holidays, vacations, and overtime.
- Adjustments for local salaries and burden rates.
- Adjustments for various software projects such as military, systems, commercial, etc.
- Support for function point metrics, lines of code (LOC) metrics, or both.
- Support for backfiring or conversion between LOC and function points.
- Support for both new projects and maintenance and enhancement projects.

Some estimating tools also include more advanced functions such as the following:

- Quality and reliability estimation.
- Risk and value analysis.
- Return on investment.
- Sharing of data with project management tools.
- Measurement models for collecting historical data.
- Cost and time-to-complete estimates mixing historical data with projected data.
- Support for software process assessments.
- Statistical analysis of multiple projects and portfolio analysis.
- Currency conversion for dealing with overseas projects.



Activities Performed	Web	MIS	Outsource	Commercial	System	Military
01 Requirements	5.00%	7.50%	9.00%	4.00%	4.00%	7.00%
02 Prototyping	10.00%	2.00%	2.50%	1.00%	2.00%	2.00%
03 Architecture		0.50%	1.00%	2.00%	1.50%	1.00%
04 Project plans		1.00%	1.50%	1.00%	2.00%	1.00%
05 Initial design		8.00%	7.00%	6.00%	7.00%	6.00%
06 Detail design		7.00%	8.00%	5.00%	6.00%	7.00%
07 Design reviews			0.50%	1.50%	2.50%	1.00%
08 Coding	30.00%	20.00%	16.00%	23.00%	20.00%	16.00%
09 Reuse acquisition	5.00%		2.00%	2.00%	2.00%	2.00%
10 Package purchase		1.00%	1.00%		1.00%	1.00%
11 Code inspections				1.50%	1.50%	1.00%
12 Independent verification and validation						1.00%
13 Configuration management		3.00%	3.00%	1.00%	1.00%	1.50%
14 Formal integration		2.00%	2.00%	1.50%	2.00%	1.50%
15 User documentation	10.00%	7.00%	9.00%	12.00%	10.00%	10.00%
16 Unit testing	30.00%	4.00%	3.50%	2.50%	5.00%	3.00%
17 Function testing		6.00%	5.00%	6.00%	5.00%	5.00%
18 Integration testing		5.00%	5.00%	4.00%	5.00%	5.00%
19 System testing		7.00%	5.00%	7.00%	5.00%	6.00%
20 Field testing				6.00%	1.50%	3.00%
21 Acceptance testing		5.00%	3.00%		1.00%	3.00%
22 Independent testing						1.00%
23 Quality assurance			1.00%	2.00%	2.00%	1.00%
24 Installation/training		2.00%	3.00%		1.00%	1.00%
25 Project management	10.00%	12.00%	12.00%	11.00%	12.00%	13.00%
Total	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**
Activities	**7**	**18**	**21**	**20**	**23**	**25**

Table 1: *Typical Software Development Activities - Six Types of Application (Data indicates the percentage of work effort by activity.)*
(Click on image above to show full-size version in pop-up window.)

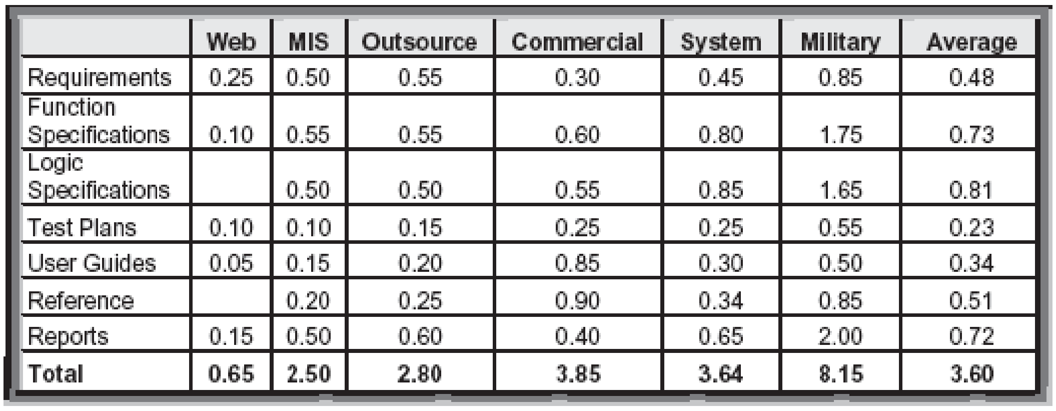
Estimates for large software projects need to include many more activities than just coding or programming. Table 1 shows typical activity patterns for six different kinds of projects: Web-based applications, management information systems (MIS), outsourced software, commercial software, systems software, and military software projects. In this context, Web projects are applications designed to support corporate Web sites. Outsource software is similar to MIS, but performed by an outside contractor. Systems software is that which controls physical devices such as computers or telecommunication systems. Military software constitutes all projects that are constrained to follow various military standards. Commercial software refers to ordinary packaged software such as word processors, spreadsheets, and the like.

Table 1 is merely illustrative, and the actual numbers of activities performed and the percentages of effort for each activity can vary. For estimating actual projects, the estimating tool would present the most likely set of activities to be performed. Then the project manager or estimating specialist would adjust the set of activities to match the reality of the project. Some estimating tools allow users to add additional activities that are not part of the default set.

Cost Drivers for Large Software Systems: Paperwork and Defect Removal

In aggregate, large software projects devote more effort to producing paper documents and to removing bugs or defects than to producing source code. (Some military software projects have been observed to produce about 400 English words for every Ada statement.) Thus, accurate estimation for large software projects must include the effort for producing paper documents, and the effort for finding and fixing bugs or defects, among other things.

The invention of function point metrics [3] has made full sizing logic for paper documents a standard feature of many estimating tools. One of the reasons for the development of function point metrics was to provide a sizing method for paper deliverables. (For additional information on function points, see the Web site of the non-profit International Function Point Users Group www.ifpug.org.)



	Web	MIS	Outsource	Commercial	System	Military	Average
Requirements	0.25	0.50	0.55	0.30	0.45	0.85	0.48
Function Specifications	0.10	0.55	0.55	0.60	0.80	1.75	0.73
Logic Specifications		0.50	0.50	0.55	0.85	1.65	0.81
Test Plans	0.10	0.10	0.15	0.25	0.25	0.55	0.23
User Guides	0.05	0.15	0.20	0.85	0.30	0.50	0.34
Reference		0.20	0.25	0.90	0.34	0.85	0.51
Reports	0.15	0.50	0.60	0.40	0.65	2.00	0.72
Total	**0.65**	**2.50**	**2.80**	**3.85**	**3.64**	**8.15**	**3.60**

Table 2: *Document Pages per Function Point for Six Application Types (Data expressed in terms of pages per function point.)*
(Click on image above to show full-size version in pop-up window.)

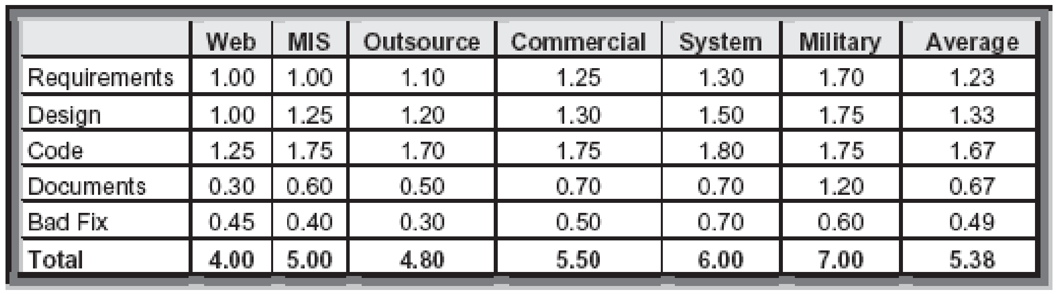
Table 2 illustrates selected documentation size examples drawn from systems, Web projects, MIS, outsource, commercial, systems, and military software domains.

At least one commercial software-estimating tool can even predict the number of English words in the document set, and also the numbers of diagrams that are likely to be present. The document estimate can also change based on paper size such as European A4 paper. Indeed, it is now possible to estimate the sizes of text-based documents in several national languages (i.e. English, French, German, Japanese, etc.) and even to estimate translation costs from one language to another for projects that are deployed internationally.

Software Defect Potentials and Defect Removal Efficiency Levels

A key aspect of software cost estimating is predicting the time and effort that will be needed for design reviews, code inspections, and all forms of testing. To estimate defect removal costs and schedules, it is necessary to know about how many defects are likely to be encountered.

The typical sequence is to estimate defect volumes for a project and then to estimate the series of reviews, inspections, and tests that the project utilizes. The defect removal efficiency of each step will be estimated also. The effort and costs for preparation, execution, and defect repairs associated with each removal activity also will be estimated.



	Web	MIS	Outsource	Commercial	System	Military	Average
Requirements	1.00	1.00	1.10	1.25	1.30	1.70	1.23
Design	1.00	1.25	1.20	1.30	1.50	1.75	1.33
Code	1.25	1.75	1.70	1.75	1.80	1.75	1.67
Documents	0.30	0.60	0.50	0.70	0.70	1.20	0.67
Bad Fix	0.45	0.40	0.30	0.50	0.70	0.60	0.49
Total	**4.00**	**5.00**	**4.80**	**5.50**	**6.00**	**7.00**	**5.38**

Table 3: *Average Defect Potentials for Six Application Types (Data expressed in terms of defects per function point.)*
(Click on image above to show full-size version in pop-up window.)

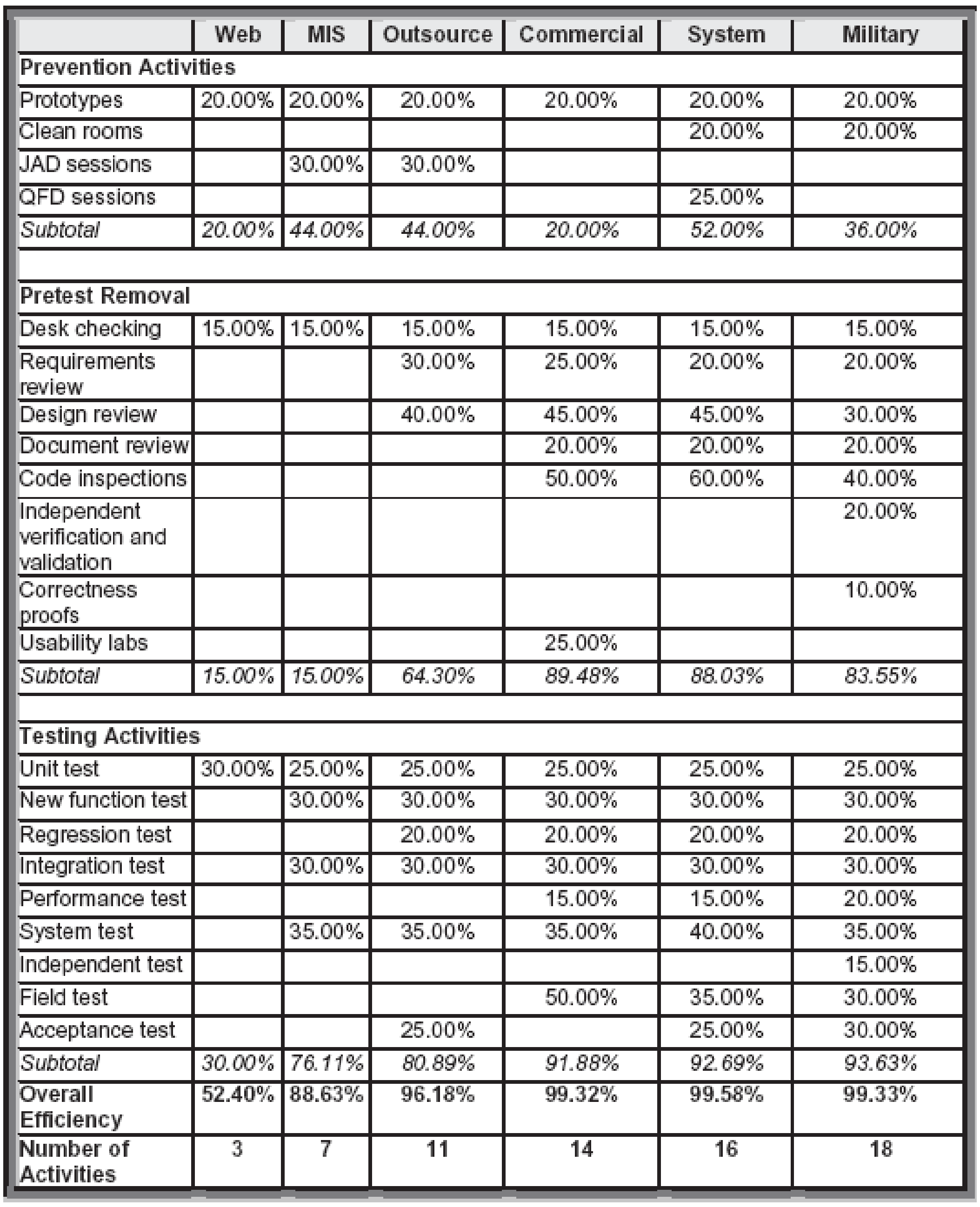
Table 3 illustrates the overall distribution of software errors among the same six project types shown in Table 1. In Table 3, bugs or defects are shown from five sources: requirements errors, design errors, coding errors, user documentation errors, and *bad fixes*. A bad fix is a secondary defect accidentally injected in a bug repair. In other words, a bad fix is a failed attempt to repair a prior bug that accidentally contains a new bug. On average, about 7 percent of defect repairs will themselves accidentally inject a new defect, although the range is from less than 1 percent to more than 20 percent bad fix injections.

The data in Table 3, and in the other tables in this report, are based on a total of about 12,000 software projects examined by the author and his colleagues circa 1984-2004. Additional information on the sources of data can be found in [2, 4, 5, 6].

Table 3 presents approximate average values, but the range for each defect category is more than 2-to-1. For example, software projects developed by companies who are at Capability Maturity Model® (CMM®) Level 5 might have less than half of the potential defects shown in Table 3. Similarly, companies with several years of experience with the Six Sigma quality approach will also have lower defect potentials than those shown in Table 3. Several commercial estimating tools make adjustments for such factors.

A key factor for accurate estimation involves the removal of defects via reviews, inspections, and testing. The measurement of defect removal is actually fairly straightforward, and many companies now do this. The U.S. average is about 85 percent, but leading companies can average more than 95 percent removal efficiency levels [7].

It is much easier to estimate software projects that use sophisticated quality control and have high levels of defect removal in the 95 percent range. This is because there usually are no disasters occurring late in development when unexpected defects are discovered. Thus, projects performed by companies at the higher CMM levels or by companies with extensive Six Sigma experience often have much greater precision than average.



	Web	MIS	Outsource	Commercial	System	Military
Prevention Activities						
Prototypes	20.00%	20.00%	20.00%	20.00%	20.00%	20.00%
Clean rooms					20.00%	20.00%
JAD sessions		30.00%	30.00%			
QFD sessions					25.00%	
Subtotal	*20.00%*	*44.00%*	*44.00%*	*20.00%*	*52.00%*	*36.00%*
Pretest Removal						
Desk checking	15.00%	15.00%	15.00%	15.00%	15.00%	15.00%
Requirements review			30.00%	25.00%	20.00%	20.00%
Design review			40.00%	45.00%	45.00%	30.00%
Document review				20.00%	20.00%	20.00%
Code inspections				50.00%	60.00%	40.00%
Independent verification and validation						20.00%
Correctness proofs						10.00%
Usability labs				25.00%		
Subtotal	*15.00%*	*15.00%*	*64.30%*	*89.48%*	*88.03%*	*83.55%*
Testing Activities						
Unit test	30.00%	25.00%	25.00%	25.00%	25.00%	25.00%
New function test		30.00%	30.00%	30.00%	30.00%	30.00%
Regression test			20.00%	20.00%	20.00%	20.00%
Integration test		30.00%	30.00%	30.00%	30.00%	30.00%
Performance test				15.00%	15.00%	20.00%
System test		35.00%	35.00%	35.00%	40.00%	35.00%
Independent test						15.00%
Field test				50.00%	35.00%	30.00%
Acceptance test			25.00%		25.00%	30.00%
Subtotal	*30.00%*	*76.11%*	*80.89%*	*91.88%*	*92.69%*	*93.63%*
Overall Efficiency	**52.40%**	**88.63%**	**96.18%**	**99.32%**	**99.58%**	**99.33%**
Number of Activities	**3**	**7**	**11**	**14**	**16**	**18**

Table 4: *Patterns of Defect Prevention and Removal Activities*
(Click on image above to show full-size version in pop-up window.)

Table 4 illustrates the variations in typical defect prevention and defect removal methods among the six domains already discussed. Of course, many variations in these patterns can occur. Therefore it is important to adjust the set of activities and their efficiency levels to match the realities of the projects being estimated. However, since defect removal in total has been the most expensive cost element of large software applications for more than 50 years, it is not possible to achieve accurate estimates without being very thorough in estimating defect removal patterns.

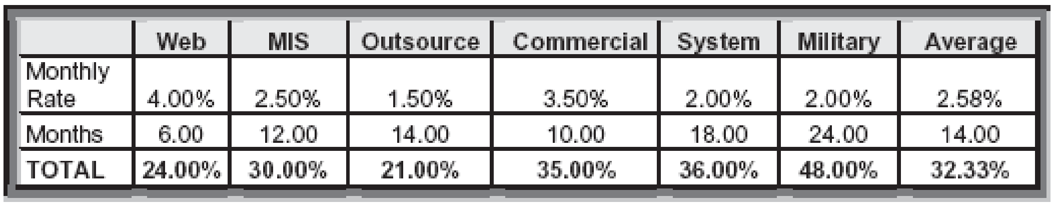
The overall efficiency values in Table 4 are calculated as follows: If the starting number of defects is 100, and there are two consecutive test stages that each remove 50 percent of the defects present, then the first test will remove 50 defects and the second test will remove 25 defects. The cumulative efficiency of both tests is 75 percent, because 75 out of a possible 100 defects were eliminated.

Table 4 oversimplifies the situation, since defect removal activities have varying efficiencies for requirements, design, code, documentation, and bad fix defect categories. Also, bad fixes during testing will be injected back into the set of undetected defects.

The low efficiency of most forms of defect removal explains why a lengthy series of defect removal activities is needed. This, in turn, explains why estimating defect removal is critical for overall accuracy of software cost estimation for large systems. Below 1,000 function points, the series of defect removal operations may be as few as three. Above 10,000 function points, the series may include more than a dozen kinds of review, inspection, and test activity defect removal operations.

Requirements Changes and Software Estimation

One important aspect of estimating is dealing with the rate at which requirements creep and, hence, make projects grow larger during development. Fortunately, function point metrics allow direct measurement of the rate at which this phenomenon occurs since both the original requirements and changed requirements will have function point counts.



	Web	MIS	Outsource	Commercial	System	Military	Average
Monthly Rate	4.00%	2.50%	1.50%	3.50%	2.00%	2.00%	2.58%
Months	6.00	12.00	14.00	10.00	18.00	24.00	14.00
TOTAL	**24.00%**	**30.00%**	**21.00%**	**35.00%**	**36.00%**	**48.00%**	**32.33%**

Table 5: *Monthly Rate of Changing Requirements for Six Application Types (From end of requirements to start of coding phases)*
(Click on image above to show full-size version in pop-up window.)

Changing requirements can occur at any time, but the data in Table 5 runs from the end of the requirements phase to the beginning of the coding phase. This time period usually reflects about half of the total development schedule. Table 5 shows the approximate monthly rate of creeping requirements for six kinds of software, and the total anticipated volume of change.

For estimates made early in the life cycle, several estimating tools can predict the probable growth in unplanned functions over the remainder of the development cycle. This knowledge can then be used to refine the estimate and to adjust the final costs in response.

Of course, the best response to an estimate with a significant volume of projected requirements change is to improve the requirements gathering and analysis methods. Projects that use prototypes, joint application design (JAD), requirements inspections, and other sophisticated requirements methods can reduce later changes to a small fraction of the values shown in Table 5. Indeed, the initial estimates made for projects using JAD will predict reduced volumes of changing requirements.

Adjustment Factors for Software Estimates

When being used for real software projects, the basic default assumptions of estimating tools must be adjusted to match the reality of the project being estimated. These adjustment factors are a critical portion of using software estimating tools. Some of the available adjustment factors include the following:

- Staff experience with similar projects.
- Client experience with similar projects.
- Type of software to be produced.
- Size of software project.
- Size of deliverable items (documents, test cases, etc.).
- Requirements methods used.
- Review and inspection methods used.
- Design methods used.
- Programming languages used.

- Reusable materials available.
- Testing methods used.
- Paid overtime.
- Unpaid overtime.

Automated estimating tools provide users with abilities to tune the estimating parameters to match local conditions. Indeed, without such tuning the accuracy of automated estimation is significantly reduced. Knowledge of how to adjust estimating tools in response to various factors is the true heart of software estimation. This kind of knowledge is best determined by accurate measurements and multiple regression of analysis of real software projects.

Summary and Conclusions

Software estimating is simple in concept, but difficult and complex in reality. The larger the project, the more factors there are that must be evaluated. The difficulty and complexity required for successful estimates of large software projects exceeds the capabilities of most software project managers to produce effective manual estimates. In particular, successful estimation of large projects needs to encompass non-coding work.

The commercial software estimating tools are far from perfect and they can be wrong, too. But automated estimates often outperform human estimates in terms of accuracy, and always in terms of speed and cost effectiveness. However, no method of estimation is totally error-free. The current best practice for software cost estimation is to use a combination of software cost estimating tools coupled with software project management tools, under the careful guidance of experienced software project managers and estimating specialists.

References

1. Jones, Capers. "Software Project Management Practices: Failure Versus Success." CrossTalk Oct. 2004 www.stsc.hill.af.mil/crosstalk/2004/10/0410Jones.html.
2. Jones, Capers. *Estimating Software Costs*. New York: McGraw Hill, 1998.
3. Albrecht, Allan. *AD/M Productivity Measurement and Estimate Validation*. Purchase, NY: IBM Corporation, May 1984.
4. Jones, Capers. *Applied Software Measurement*. 2nd ed. New York: McGraw Hill, 1996.
5. Jones, Capers. *Software Assessments, Benchmarks, and Best Practices*. Boston, MA: Addison Wesley Longman, 2000.
6. Kan, Stephen H. *Metrics and Models in Software Quality Engineering*. 2nd ed. Boston, MA: Addison Wesley Longman, 2003.
7. Jones, Capers. *Software Quality - Analysis and Guidelines for Success*. Boston, MA: International Thomson Computer Press, 1997.

About the Author

Capers Jones is founder and chief scientist of Software Productivity Research (SPR) LLC. He has almost 40 years of experience in software cost estimating. Jones designed IBM's first automated estimation tool in 1975, and is also one of the designers of three commercial software estimation tools: SPQR/20, Checkpoint, and KnowledgePlan. These software estimation tools pioneered the use of function point metrics for sizing and estimating. They also pioneered sizing of paper documents, and the estimation of quality and defect levels. To build these tools, SPR has collected quantified data from more than 600 companies.

Software Productivity Research, LLC
Phone: (877) 570-5459
E-mail: cjones@spr.com