Determining software schedules

Capers Jones
Software Productivity Research

Improving software productivity, shortening schedules or time to market, and improving quality are prominent topics in software journals in both contributed articles and advertising copy. Unfortunately, most of these articles and advertisements have dealt with software schedules, productivity, or quality factors in abstract terms such as "Buy our tool and improve productivity by 10 to 1%" or "Cut development schedules by 50 percent." But now we can measure these factors with reasonable accuracy and collect empirical data on both "average" and "best-in-class" results. We are particularly interested in the wide performance gaps between laggards, average enterprises, and industry leaders, as well as differences among the various software domains. The function-point metric lets us establish a meaningful database of software performance levels. A simple algorithm raises function points to a total to obtain a useful first-order schedule estimate.

Beginning and end points of software schedules

Considering how important software schedules and time-to-market considerations are, it is surprising that in the past 50 years, this topic has inspired very little solid, printed data. Although many commercial-software cost-estimating tools can predict schedules with fairly good accuracy, as of 1994 only about 15 percent of US software managers—and even less abroad—were using them. So at least 85 percent of the world's software managers jump into projects with hardly a clue as to how long they will take. This explains why about half the major disasters associated with missed schedules and overruns can be traced back to the schedules being incompetently established in the first place.

From collecting schedule information on historical projects, I've observed that establishing a software project's true duration schedule is one of the trickiest measurement tasks in the entire software domain. A software project's shipping date is usually fairly clear, but the date the project actually originated is the hardest and most imprecise data point in the software industry. For most software projects, there is an amorphous, unmeasured period during which clients and software personnel grapple with fundamental issues such as whether to build or buy, and what is really needed. Of several thousand software projects that my colleagues and I have analyzed over the past 10 years, less than 1 percent had a clearly defined starting point. To avoid this ambiguity, our pragmatic approach is to ask the project's senior software manager to pick a date as the starting point and simply use that. If the project has formal, written requirements, we can sometimes ascertain the date they were introduced, or at least the date shown on the first printed version.

Surprisingly, more than 15 percent of software projects are also ambiguous as to when they were truly delivered. One source of ambiguity is whether to count the start of external beta testing as the delivery point or to wait until the formal delivery when this testing is over. (Our general rule is to count the first formal release to paying customers or users that are not participants in beta testing or prerelease field testing.) Another source of ambiguity is whether to count a software product's initial delivery or to wait until deferred functions come out a few months later in what is usually called a point release (such as "Version 1.1").

Not only are both ends of software projects ambiguous and difficult to determine, but the middle can get messy, too. Even with the "waterfall model" of development, there is always overlap and parallelism between adjacent activities, so that a project's end-to-end schedule is never the same as the duration of those activities. Software requirements are usually only about 75 percent defined when design starts. Design is often little more than 50 percent complete when coding starts. Integration and testing can begin when coding is less than 20 percent complete. And user documentation usually starts when coding is about 50 percent finished. Moreover, the newer spiral and iterative models of software development are even more amorphous, since requirements, design, coding, testing, and documentation can be freely interleaved and take place concurrently.

Space does not permit a full discussion of all the factors that affect software schedules or of the schedule durations...
for all kinds of software projects. But the function-point metric provides a convenient, quick estimator for schedule durations that can be applied early in a software project’s development cycle. The schedule interval runs from the project’s nominal “start of requirements” to the “first customer ship” date. The activities in this interval include requirements creation, analysis, high-level and detail design, reviews and inspections, coding, integration, all forms of testing, and user-document creation.

Once the application’s function-point total has been determined, it is only necessary to raise that total to the power shown in Table 1. For example, an “average” military project of 10,000 function points (equivalent to about 700,000 Ada statements) would use a power of 0.46. Raising 10,000 function points to the 0.46 power, we would estimate the project’s nominal schedule at 69.18 calendar months. (Note that for enhancement schedules, we would use the enhancement’s—not the original product’s—function-point total as the estimate basis. Thus, if we added 100 function points to a 5,000-function-point system, we would use 100 as the starting point.) This simplistic methodology is not a substitute for a true software-estimating tool that can deal with factors such as programming languages, tools, reuse, experience levels, creeping requirements, and other characteristics known to influence a software project’s outcome. But it does enable a useful first-order approximation with any pocket calculator that can handle power calculations. In fact, clients who have used this methodology have found that it works reasonably well. However, leading-edge companies whose available volume of reusable material exceeds 50% can surpass even the “best in class.”

Software schedules in different domains

Software schedules in the four domains—commercial software, management information systems (MIS), systems software, and military software—are strongly impacted by several considerations.

Commercial software. This domain includes products marketed to customers, such as spreadsheets, word-processors, and accounting packages. The average size of modern, windows-based commercial products for typical word-processing and spreadsheet packages is roughly 1,000 function points. However, large mainframe software packages, such as order-entry systems, databases like IBM’s DB2 or IMS, and manufacturing-support packages, can exceed 10,000 function points. The primary schedule consideration in the commercial world is quality, and the reason most commercial software projects run late is that they encounter too many bugs during testing to be released.

Management information systems. MIS concerns custom software that companies build for their own use or that they commission contractors or outsource vendors to build for them. An MIS project’s size ranges from less than 100 function points to more than 20,000 for major systems involving insurance-claims handling, manufacturing control, or full order-entry systems.

The MIS domain’s two key schedule drivers, quality and creeping user requirements, tend to interact in unpleasant ways. The rate of unanticipated user-requirement changes in a typical MIS project is about 1 percent per month. So for a three-year project, more than a third of the final functionality may be added midstream. Many MIS shops are not very sophisticated in software quality control and omit important activities such as reviews and inspections. Since quality control on creeping user requirements is even worse than normal, lax development methods coupled with volatile requirements result in many MIS projects running amok around the time testing starts. If the projects are developed carelessly, which unfortunately is the case for many new client/server applications, the resulting poor quality may be passed on to users, because testing is often perfunctory.

Systems software. This domain involves software that controls physical devices such as telecommunication-switching, fuel-injection, flight-control, and operating systems. Software ranges from about 250 function points to more than 50,000 function points for huge systems such as large mainframe operating systems. Because quality levels are high in the systems-software domain, the better systems producers are in a good position to take advantage of reusability and increase the parallelism of their development methods. In fact, some projects are setting new records and improving the “best-in-class” results.

The systems-software community has learned that full quality control, including formal pretest inspections, is necessary to do business. This also implies formal specifications, so the important schedule factors for systems software are creating and reviewing a significant volume of specification material and executing an extended series of defect-removal operations. A typical defect-removal cycle for a large system-software project, such as a telephone-switching system of around 10,000 function points, would include formal inspections of design, code, and documents, and at least seven kinds of testing: unit, new-function, regression, integration, stress, system, and field tests.

Military software. Applications that are required to follow US military standards such as DoD 2167A are included in this domain. Military projects range from those of perhaps 200 function points to the largest software systems in human history, which can exceed 250,000 function points. Military software schedules have four key gating factors: the initial procurement process, enormous volumes of paperwork created in the DoD community, high quality levels for mission-critical applications, and

<table>
<thead>
<tr>
<th>Software domain</th>
<th>Best in class</th>
<th>Average</th>
<th>Worst in class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commercial</td>
<td>0.39</td>
<td>0.42</td>
<td>0.45</td>
</tr>
<tr>
<td>MIS</td>
<td>0.41</td>
<td>0.43</td>
<td>0.46</td>
</tr>
<tr>
<td>Systems</td>
<td>0.43</td>
<td>0.45</td>
<td>0.48</td>
</tr>
<tr>
<td>Military</td>
<td>0.45</td>
<td>0.46</td>
<td>0.50</td>
</tr>
</tbody>
</table>
the special activities that only military projects perform. Software (and hardware) is typically procured via external contracts derived from various bids and proposals. This process is both intrinsically time-consuming and rife with challenges and litigation from disgruntled vendors who have not received the contract. Thus, a typical military software project takes about 35 percent longer to get started than a similar civilian project. If the initial procurement leads to major litigation, the schedules can be even longer.

US military standards have been useful in many ways, but they have had some unintended side effects. One of the most significant is the enormous volume of specifications and planning documents. By actual count, military software projects that follow the DoD 2167A standard have averaged about 400 English words for every Ada statement. This is about three times more than civilian norms and twice as much as the second-place paperwork-generating industry (telecommunications).

Quality control for military projects is generally the same as for civilian systems software and includes formal inspections followed by a full suite of testing steps. But military projects also tend to perform many activities that are essentially unique and have no equivalence in the civilian domain. One notable example is an IV&V (independent verification and validation), a structured review of plans and specifications by a contractor different from the primary contractor. The value of this rather time-consuming undertaking has not been formally explored.

In mid 1994, the Secretary of Defense stated that US military organizations should begin to move toward civilian best practices and away from the somewhat baroque, convoluted defense approaches and military standards that have developed and accreted over the past 20 years. This is a promising concept, but it is too soon to judge whether it will prove effective.

Because military quality levels are fairly high, the military domain is in a position to increase material reusability and perhaps begin to match the civilian sector in schedule results. It remains to be seen whether this will really occur. But at least the military services are exploring civilian best practices, even if they don’t fully adopt them.

Schedules and time-to-market intervals are among the most important factors in the software industry. For more than 30 years, there has been a shortage of empirical data and useful rules for estimating software schedules early and accurately. The function-point metric is not a panacea, but it does provide a first-order approximation that can easily be carried out on a pocket calculator. Since function points are derived during requirements and early design, the ability to estimate schedules before committing major resources to a project is a practical way to minimize later problems.

---

**FCCM’95**

**IEEE SYMPOSIUM ON FPGA's FOR CUSTOM COMPUTING MACHINES**

Napa Valley, California  April 19-21, 1995

**CALL FOR PARTICIPATION**

Participation by researchers is solicited. Papers will be presented on the use and application of field programmable gate arrays, field programmable interconnects, and related devices as computing elements in attached or special-purpose processors or coprocessors, especially including, but not limited to, coprocessor boards for augmenting instruction sets of general-purpose computers; attached processors for specific purposes or general purposes; languages for programming FPGA-based computers; tools and environments for programming FPGA-based computers; application domains suitable for FPGA-based computers; architecture emulation using FPGA-based computers; teaching computer architecture with FPGA-based computers. A special session will feature vendors of hardware and software presenting new or upcoming products involving FPGAs for computing. This conference is sponsored by the IEEE Computer Society and the TC on Computer Architecture. The proceedings will be published by the IEEE Computer Society.

The symposium will be held at the Marriott at Napa Valley, 3425 Solano Drive, Napa, California 94558, (707)253-7433 voice, (707)258-1320 FAX. Reservations for accommodations should be made directly with the hotel. A special group rate is available to FCCM’95 attendees. Registration information and other information related to the symposium is available on the WWW:

URL (MOSAIC) PAGE:  [http://www.super.org:8000/FPGA/comp.arch.fpga](http://www.super.org:8000/FPGA/comp.arch.fpga)

Further details can be obtained from either of the two conference co-chairs:

- Peter Athanas, Virginia Tech, Department of Electrical Engineering, 340 Whitemore Hall, Blacksburg, Virginia 24061-0111. (703)221-7010 voice (703)221-3362 faxes, athanas@vt.edu
- Kenneth Pocek, Intel, Mailstop RN6-18, 2200 Mission College Blvd., Box 58119, Santa Clara, CA 95052 (408)765-6705 voice (408)765-5165 faxes, kpocek@sc.intel.com

Organizing Committee:
- Jeffrey Arnold, Supercomputing Research Center
- Pak Chan, Univ. California, Santa Cruz
- Brad Hutchings, Brigham Young Univ.
- Wayne Luk, Imperial College of London

Duncan Buell, Supercomputing Research Center
Apostolos Dollas, Technical Univ. of Crete
Tom Kean, Xilinx, Inc. (U.K.)

February 1995