The effects of the business model on object-oriented software development productivity

by T. E. Potok
M. A. Vouk

Unless the business model that governs software production adjusts to new technology, it is unlikely that an investment in the technology will result in real productivity benefits. Commercial development always takes place in the context of a business model, and in that context an understanding of how business constraints influence commercial software development is imperative. As software markets become more competitive and business pressures shorten software development cycles, improved software development productivity continues to be a major concern in the software industry. Many believe that new software technology, such as object-oriented development, provides a breakthrough solution to this problem. Unfortunately, there is little quantitative evidence for this belief. In this paper we explore the relationship between the business model and the productivity that a software development methodology can achieve in a commercial environment under that model. We first examine empirical data from several commercial products developed using object-oriented methods. The results indicate that object-oriented development may not perform any better than "procedural" development in environments that lack incentives for early completion of intermediate project tasks. We then model and simulate the impact of the software task-completion incentives and deadlines on the productivity that might be expected from a technology with high-performance potential. We show how and why some common business practices might lower project productivity and project completion probability. We also discuss to what extent poor software process control and (im)maturity of the technology compounds the problem.

It is widely believed that object-oriented (OO) development[1] has considerable potential for increasing software development productivity. The reasons for the gains range from reuse, through better problem understanding, to better (less complex and less costly) designs and implementations. However, there is little quantitative evidence that productivity of real-life object-oriented software development is indeed consistently better than that of "classical" or "procedural" software development. Furthermore, most studies related to object-oriented development productivity do not consider it in conjunction with the business practices under which the software is being developed. According to Jacobson et al.:

> A business model shows what the company's environment is and how the company acts in relation to this environment. By environment we mean everything the company interacts with to perform its business processes, such as customers, partners, subcontractors and so on. It shows employees at every level what must be done and when and how it should be done.[2]

0018-8670/97/$5.00 © 1997 IBM

Business models tend to focus on cost and calendar events (e.g., quarterly reports) and tend to form deadlines that are governed by marketing and competitive pressures, often regardless of the real software engineering capabilities of the organization. On the other hand, software engineering development models tend to focus mainly on the complexity of a software project and the capabilities of the development teams and the software methodologies. In a meaningful evaluation of project viability we need to consider both in an integrated fashion.

In this paper we model and quantitatively explore the relationship between the business incentives and deadlines, and the productivity that a potentially high productivity software development methodology can achieve in a commercial environment. We use object-oriented software development as a specific example of such a technology, and we develop our models and analyses based on empirical information we have collected about object-oriented development as practiced in a commercial development environment.

Related work. Lewis et al. performed an experiment with undergraduate software engineering students to study the effects of reuse.[3] Based on their tests of the recorded productivity metrics, they concluded that the object-oriented paradigm can improve productivity by about 50 percent when reuse[4] is present. However, they did not find any statistically significant evidence that the object-oriented paradigm has higher productivity than procedural methods when reuse is not a factor. Melo et al. conducted an experiment with graduate students that resulted in seven projects ranging in size from 5000 to 25 000 lines of code.[5] The projects were developed using a waterfall process model, object-oriented design, C++, and varying levels of reuse. Their results support the conclusion that reuse rates can increase programmer productivity as much as two to three times.[6] In general, optimistic economic models of reuse indicate that break-even reuse levels may be as low as 10–20 percent,[7] while pessimistic models show that break-even levels may be difficult to achieve even under very high levels of reuse.[8]

There is also evidence that different development methodologies have differing effects on the software development process. For example, Boehm-Davis et al. report on a comparison of Jackson program design, object-oriented design, and functional decomposition, using professional programmers.[9] Some of the insights from the study are that Jackson's method and object-oriented methodologies produce more complete solutions, require less time to design and code a problem, and produce less complex designs than functional decomposition. However, a quantitative comparison of productivity associated with different methodologies was not given. Zweben et al., again in an experiment with graduate and undergraduate students, show that in Ada layering and encapsulation (an object-oriented trait) may reduce development effort.[10]

There are many other studies and books that describe the benefits of the object-oriented approach in general[11–18] and of the value of reuse in particular.[7,19–23] However, there are very few, if any, convincing quantitative studies that focus on the productivity related to software developed for commercial use by professional programmers who use object-oriented methods. A recent paper by Hansen indicates that commercial software development should always be considered in the context of its business model.[24] Our own work, discussed in more detail in later sections, supports this.[25,26]

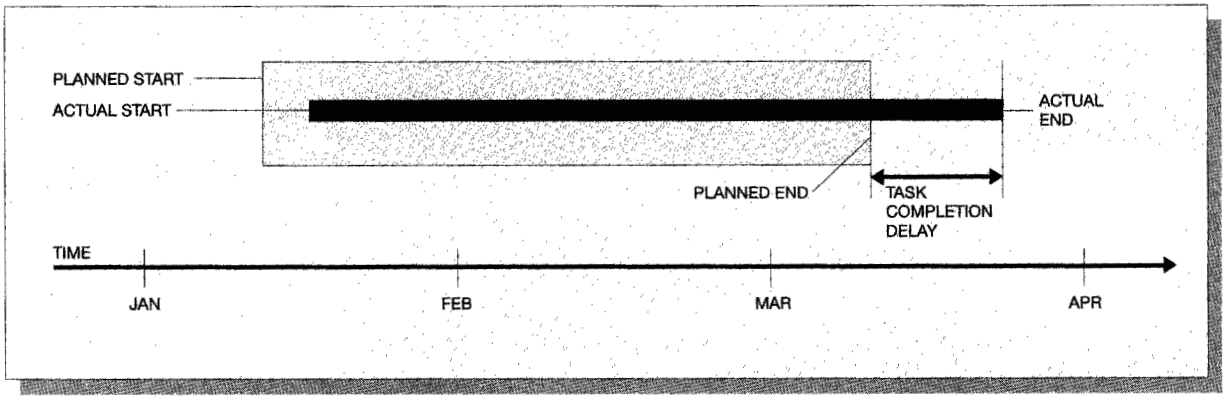
Approach. In this study we focus on the effects that some business practices may have on the productivity observed in software projects. We used empirical information to identify the effects and to help formulate a detailed simulation model of interactions among an iterative software development process, its maturity, and the applied business model. We then used this model to explore how business constraints can affect productivity and market timeliness when potentially high-productivity methodologies, such as object-oriented development, are used.

In the next section we present some empirical data that relate business practices and software development productivity. In a later section we formulate a simulation model of the interactions, then use the model to study the impact of business-imposed incentives and deadlines on the software development productivity that might be expected from a relatively new high-performance software technology. Summary and conclusions are given in the last section.

Empirical information

We began our study with empirical information obtained from a large commercial software development organization. We describe the information in the vocabulary of the organization, and we start this section by defining the terms needed to do so. We also define the metrics we used in our analysis.[27]

Figure 1 Task completion delay



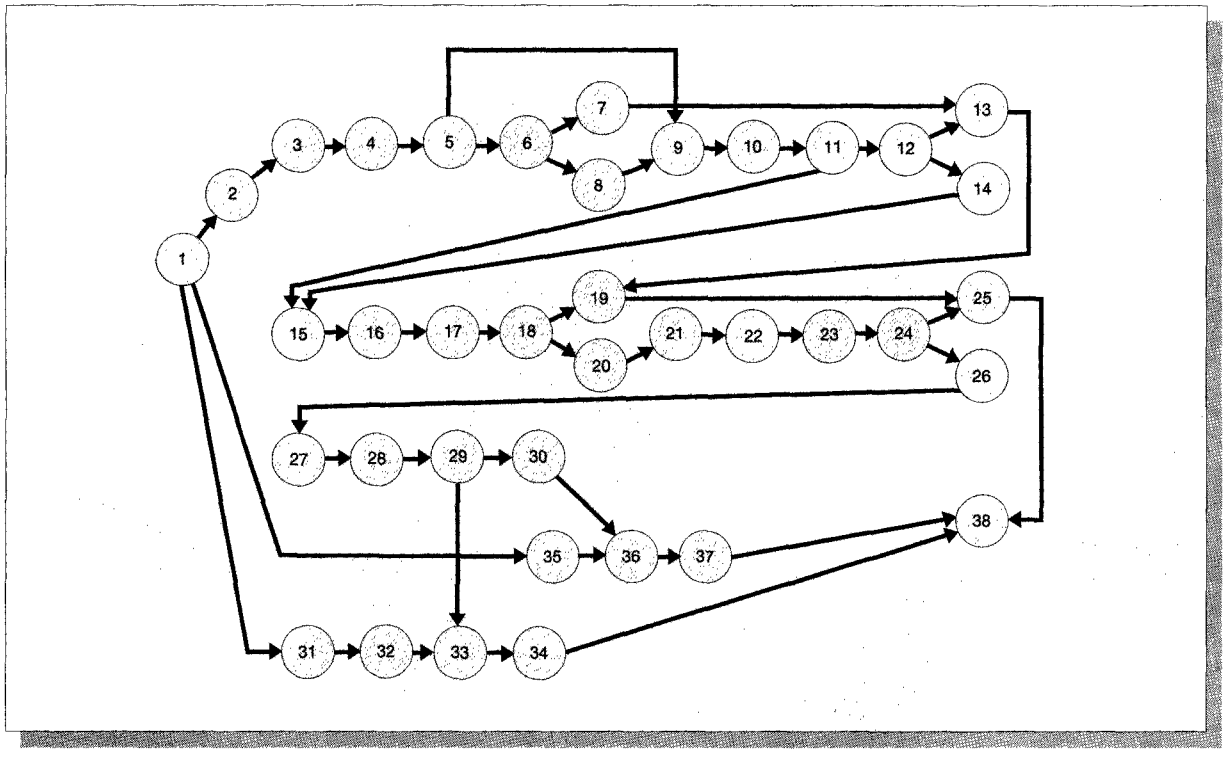
Metrics and definitions. In the context of this paper we define productivity in terms of new and changed product lines of code (LOC), but with an understanding that the effort (or time) expended includes many activities in addition to coding that are necessary for developing a viable commercial product. A software team may consist of one or more software professionals, not all of whom need to be engaged in software coding and testing activities. We define *average individual productivity* of a software professional on the team in LOC per person-month. In contrast, to focus on the *calendar-time* nature of the marketing requirements and other business-related factors, we express software *team productivity* in terms of LOC or thousands of LOC (KLOC) per calendar month.

A *software product* is a commercially available software system that includes packaged software, documentation, and support. It is thoroughly tested, and its quality is certified prior to release. It can be developed using classical, or procedural[28] methods or object-oriented methods. In general terms, each delivery of a software product is considered a *generation*. Specifically, a software product is referenced by *version* and *release*. We will refer to second and later versions or releases as *follow-on* versions or releases. A software *product schedule* directs the execution and completion of a series of tasks, from the initial planning stages through the final product shipment. A *task* or *activity* is a unit of work that requires a finite amount of time to complete. *Individual tasks* can be viewed as individual segments of a project. The significant events in the schedule are called *milestones*. In the context of an iterative process (such as the one described in the next subsection), we distinguish *project iterations* as collections of individual tasks.

In our experience, the statistic that best highlights the influence of business-related factors on software development tasks is the completion of tasks relative to a planned deadline. We call it *task completion delay* (see Figure 1). This value is the difference between the planned end date and the actual end date for a given task. A negative value indicates that the task finished early, a zero value indicates the task was on schedule, and a positive value shows that the task finished late. We use this variable to eliminate differences in individual task start and completion calendar dates when we compare tasks as a population.

Software and business processes. The empirical data were collected at the IBM Software Solutions Laboratory in Research Triangle Park, North Carolina. The laboratory was certified ISO 9000[29] in 1994, and it has consistently received high marks in internal assessment against the Malcolm Baldrige[30] criteria. The general model that drives its software development recognizes two major software product subcategories: *versions* and *releases*. A new version is typically quite large and contains a significant product enhancement, or change in functionality. A version is ordinarily followed by one or more maintenance releases, usually much smaller than a version, that contain defect fixes and minor enhancements. The calendar-time duration for the development of versions and releases is strongly driven by market forces. Versions tend to take longer than releases, but are within the 18- to 24-month time frame common to

Figure 2 The development process used for a second-generation object-oriented project



the industry today.[31] Release development will normally be at least 9 to 12 months. Reasons for this include distribution costs, arrival rate of release-type fixes and changes, and even user-perceived quality (e.g., scheduling a release very soon after a version can give the impression of quality problems). While all new development must be completed with a limited number of personnel, existing projects will have an established team. Typically an effort is made to maintain (or even increase) the size of the team, because it may not be cost-effective to dismantle it between versions. Therefore, it is not unusual to have a large version developed with tight resource and time constraints, then a smaller follow-up maintenance release developed over a more relaxed schedule using the same team.

The development of both versions and releases is subject to frequent high-level reviews of their status against dates for key development milestones established at the beginning of the product cycle. The progress toward these dates is reviewed regularly and in detail, and schedule slips against any major milestones are strongly discouraged. Detailed project schedules are required at the beginning of the product development cycle, and they trigger business processes, including funding, planning, marketing, supporting, and certification of the quality of a product. The most prevalent software development process followed in the organization is called "iterative." The iterative process is a variant of a combination of evolutionary prototyping[32] and "successive versions."[33] In theory, each software "iteration," is fully planned, designed, coded, and tested before work begins on the next iteration.[34] The duration and amount of code produced for each iteration is approximately the same. A typical project activity diagram is illustrated in Figure 2.

Figure 2 shows a high-level PERT (program evaluation and review technique) diagram of the process used for one of the commercial products developed at the laboratory. The product was a second-generation object-oriented "port" between platforms. In this diagram, edges represent activities and have durations associated with them, while nodes are mile-

Table 1 Description of activity nodes shown in Figure 2

Node	Edge	Description	Node	Edge	Description	Node	Edge	Description
1	1–2	Project start	12	12–13	Unit test		24–26	
	1–31			12–14		25	25–38	Update analysis model
	1–35		13	13–19	Update analysis model	26	26–27	Review iteration
2	2–3	Define analysis model	14	14–15	Review iteration	27	27–28	Plan iteration
3	3–4	Plan iteration	15	15–16	Plan iteration	28	28–29	Define design model
4	4–5	Define design model	16	16–17	Define design model	29	29–30	Code
5	5–6	Code	17	17–18	Code		29–33	
	5–9		18	18–19	Unit test	30	30–36	Review iteration
6	6–7	Unit test		18–20		31	31–32	Function test prep
	6–8		19	19–25	Update analysis model	32	32–33	Define test plan
7	7–13	Update analysis model	20	20–21	Review iteration	33	33–34	Function test
8	8–9	Review iteration	21	21–22	Plan iteration	34	34–38	Test report
9	9–10	Plan iteration	22	22–23	Define design model	35	35–36	Customer validation preparation
10	10–11	Define design model	23	23–24	Code	36	36–37	Customer validation
11	11–12	Code	24	24–25	Unit test	37	37–38	First customer ship
	11–15					38		Project end

stones. Different activities and milestones are described in Table 1. The final product delivered approximately 64 000 of C++ code; the port required over 8 person-years of effort and took 16 months to complete. A Booch-type[11] object-oriented methodology was used.

There are five (unfolded) iteration cycles. The first iteration ends with milestones 7 and 8, the second with 13 and 14, the third with 19 and 20, the fourth with 25 and 26, and the final iteration with node 30. The system testing activities run in parallel but are mainly focused on the software emerging out of the final cycle.[35] When an iteration is completed the work is reviewed, and suggested changes and enhancements are examined during the planning phase of the next iteration. When all development iterations are completed, depending on the measured product quality, the product may be ready for delivery or it may require additional system testing.

Previous results. We examined 19 commercially available software products from the IBM Software Solutions Laboratory. There were three distinct categories of products, those developed using procedural methods, those developed using object-oriented methods, and those developed using object-oriented methods and later ported to another platform. Four projects were ports, seven were developed using object-oriented methods, and eight were developed with procedural methods. All object-oriented efforts were either first- or second-generation products. The details of that study are reported elsewhere.[25,26,36] In the following paragraphs we briefly summarize the key findings. We found:

1. No statistically significant difference between software development productivity recorded for procedural and object-oriented products that were not ports
2. An unusual economy of scale for both object-oriented and procedural software development that was difficult to explain with traditional productivity factors
3. A remarkably compliant tracking pattern between the actual and planned deadlines of several project schedules that we examined in detail

While software ports are expected to exhibit higher productivity,[37] it was surprising to see that, on the average, there was no significant difference in productivity between object-oriented and procedural software development. This contradicts studies done on object-oriented productivity in a noncommercial environment, leading to the question of the business influences over object-oriented development. The data also show a very strong economy of scale. For example, the time per programmer required to complete a 1 KLOC task decreased as project size increased, and as a general rule, smaller projects exhibited lower productivity than larger projects regardless of the methodology. This result, although not unique, counters most previous studies.[32] Further, there is no obvious explanation as to why this result has occurred. It is possible that there is a large but constant overhead associated with all projects, or that smaller projects are, for some reason, more complex, but there was no evidence of either. However, there was evidence[38] that larger teams may have been dictated for certain types of smaller projects, namely intermediate product releases, to preserve

continuity of skills and expertise between large releases of the product. While this could provide a partial explanation for lowered productivity in the smallest projects, it does not really explain the productivity growth observed for larger projects. One could also argue that larger projects have stronger development teams, accounting for the economy of scale. This also was not supported by our data. Based on interviews and observations, the development teams had roughly the same experience and skill level throughout the organization.

The remarkable schedule adherence between the actual and planned deadlines that we observed was also difficult to explain solely on the basis of the software methodology. The deadlines are used by management, in accordance with the defined business processes and culture, to control product development and delivery, so one possible explanation is that the product planning process was very accurate. But, given the very wide variation in the observed average productivity over the examined projects, it is more likely that good schedule compliance was achieved through dynamic schedule enforcement of deadlines for key milestones than that it was achieved using highly accurate productivity forecasting. Another explanation could be that the schedules were met because software functionality was changed or testing time was reduced to meet them. Examination of the project records showed that no major functions were added or deleted in these projects, and that time was not saved by shortening testing cycles.

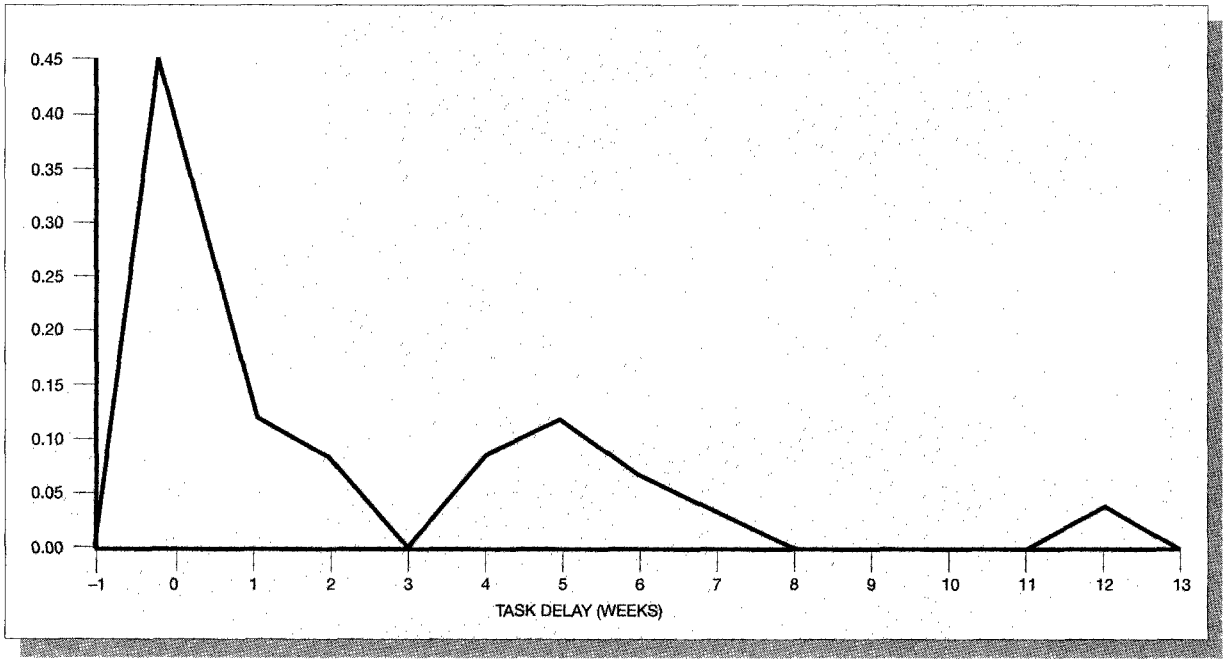
This information prompted us to conjecture that the governing influence may be, not the software development technology, but the business model, which includes market and business constraints imposed on schedules, tasks, and resources. It also prompted us to hypothesize that two related effects, Parkinson's Law[39,40] and the Deadline Effect,[32,41] are probably the key factors in many commercial software development efforts. Parkinson's Law states that work will expand to fill the allocated time. For example, if a project is assigned to three similar development teams with three easily achievable, but different deadlines, the projects will not complete at the same time, but according to the deadlines set. The Deadline Effect occurs when programmers are compelled to invest extra effort in order to complete a task by a given deadline. If there is strong pressure to meet a deadline, people will work additional hours solely to meet the deadline. We consider these effects as special cases of "goal theory" described in industrial psychology literature. Industrial psychologists report[42] significant evidence that personal productivity increases with specific, challenging goals, such as aggressive deadlines.[43,44] This supports the notion that programmer productivity can be strongly influenced by schedule goals. For example, given the same task size and complexity, team productivity for hard, specific schedules will most likely be higher (within reason) than it is for less challenging schedules.

Parkinson's Law and the Deadline Effect are probably the key factors in many commercial software development efforts.

To illustrate how Parkinson's Law and the Deadline Effect apply in our case, and that the task completion schedule compliance we observed is probably the result of a combination of process and business factors, we now present an analysis of task delay data for three project schedules, one for each project category we encountered.

Task completion delay analysis. All products were developed under the iterative process model described earlier, and under the same business model. The project subset discussed here includes: a first-generation product version with 38 KLOC in C++ developed using object-oriented methods (Project A), a second-generation product version that was a port of 64 KLOC in C++ also developed using object-oriented methods (Project B, shown in Figure 2), and a follow-on maintenance release of a product where the development team followed a procedural methodology to produce 6 KLOC in C (Project C). In addition to the variation in product sizes, the projects showed wide variation in average individual productivity. The procedural project was lowest, the first-generation object-oriented project was two and a half times greater, and the ported object-oriented project was nearly ten times greater than the first-generation object-oriented project. Although we hope that at least some of the credit for these differences in productivity can be given to the object-oriented

Figure 3 Task completion delay distribution for Project A (first-generation object-oriented project)



methodology, other factors can produce similar variability.

For each major task of the three projects, we obtained the following raw scheduling information: the planned start time, the planned end time, the actual start time, and the actual end time. The planned end time represents the task completion deadline. Each task may have its own deadline, or there may be a common deadline for several tasks. The granularity of the task schedules was typically from one to four weeks and involved from one to three software professionals. Examples of tasks are design-level reviews, the design of a small component, and the unit test of a component (see Table 1). From the raw data we can calculate a number of task parameters, such as the planned task duration, the actual task duration, early task starts, late task starts, and so on.

In Figures 3, 4, and 5 we plot histograms of individual task completion delays (rounded to the nearest week) for the three projects. While all three met the original planned shipping deadline, the distribution of task completion delays shows that a number of dates for intermediate milestones were not met. For example, the plot in Figure 3 shows the fraction of tasks with a given completion delay (in weeks) for Project A. The distribution has a peak at zero, indicating that about 45 percent of the tasks required to develop this project finished on the planned deadline (note that all tasks planned for the final shipping date are in this category). However, the remaining 55 percent of the tasks missed the planned deadlines. For instance, the secondary peak around Week 5 is due to a four-to-five week slip in several of the coding and driver-build tasks, while the third peak at Week 12 was due to administrative delays in getting the design specification approval signatures.

Figure 4 shows the completion delay distribution for Project B. This distribution shows that most (over 60 percent) of the tasks finished on schedule. Again, all tasks scheduled for the final (aggregate) deadline were in this category. However, while this team had the highest productivity level of the three, only one task finished early.

Finally, Figure 5 shows the task completion delay distribution for Project C. We again see that most of the tasks completed on time. This includes all tasks scheduled for the final deadline. The fraction of on-

Figure 4 Task completion delay distribution for Project B (second-generation ported object-oriented project)

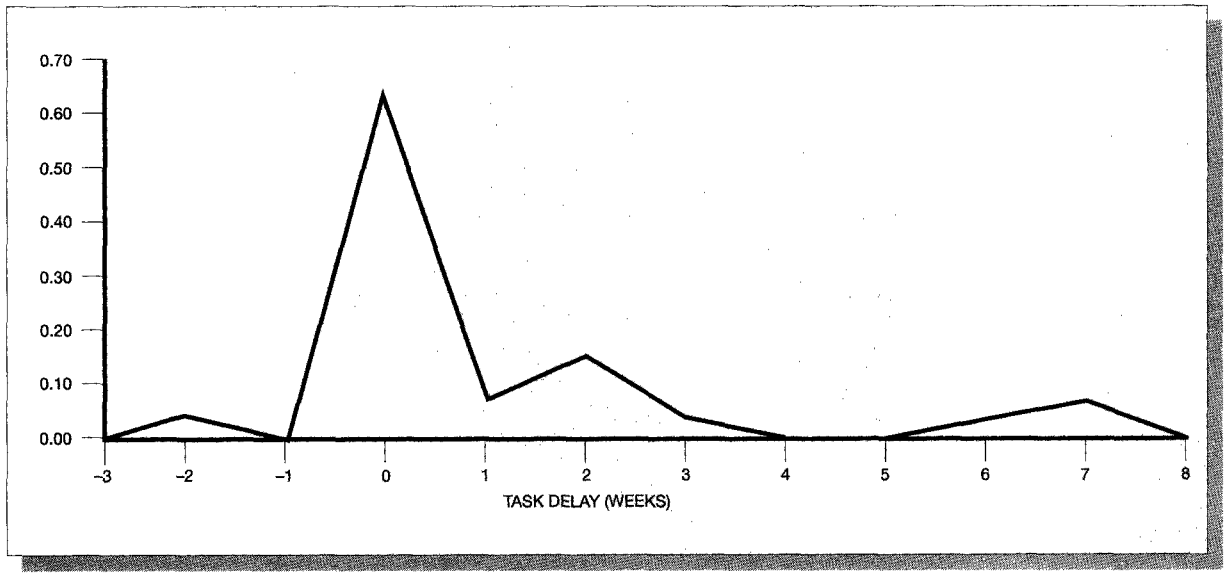
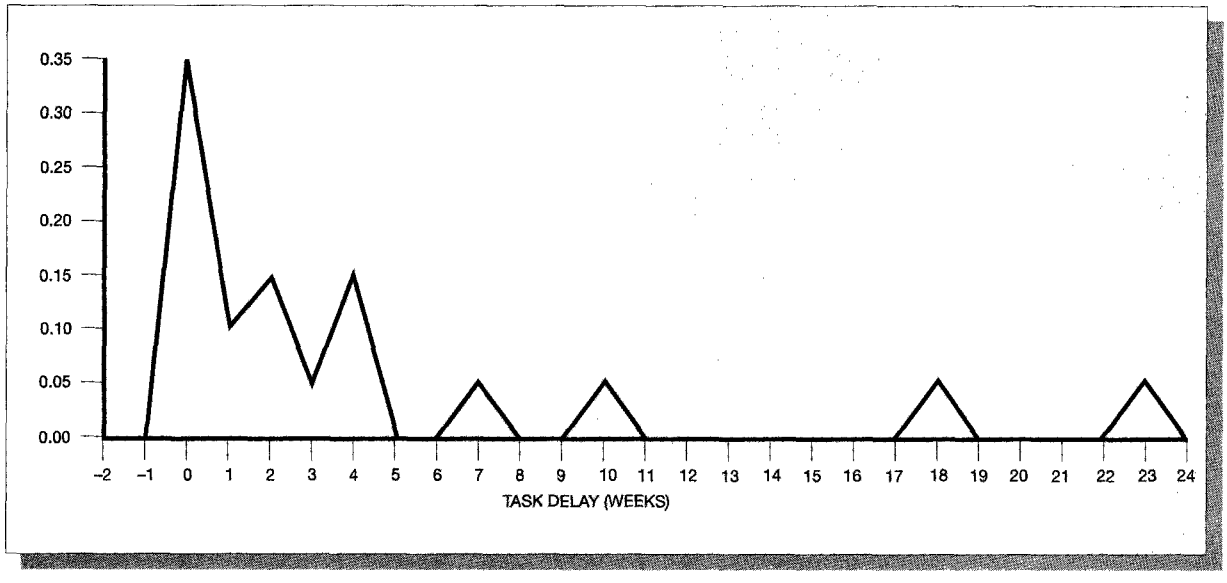


Figure 5 Task completion delay distribution for Project C (procedural project)



time tasks was smaller than in the other two projects, and there is a much larger range in the delay distribution. Some intermediate tasks were as many as 23 weeks late.

We see that although the three projects are quite different in nature, they show some interesting similarities that might be expected, given the business model under which they were developed. Specifically:

1. In all three projects the most frequent value for the task completion delay was zero. About 35 to 60 percent of the tasks finished on the date originally planned. This includes *all* tasks scheduled

for the final, aggregate project deadline. It is an indicator that a very strong mechanism for enforcement of the final deadline, and of many key intermediate deadlines, was in effect.
2. Apparently, it is uncommon to finish a task early. Only one project showed a task completing early. This is a strong indicator that Parkinson's Law was operating.
3. In all three cases, a (small) group of intermediate- or low-priority tasks was significantly late, from 7 to 23 weeks after the original deadline.

The business model used to guide these and other projects that we examined strongly discourages late completion of key milestones, since the final deadlines tend to be driven by market pressures. Failure to meet a key deadline usually has strong negative consequences. However, the same model does not provide strong incentives for *early* completion of either intermediate or final milestone tasks. Indeed, examination of the significantly late tasks revealed that these tasks were not only of low priority, they did not in any way gate (restrict) the development of their product.

During our task completion delay analysis, we found confirmation of another of our previous results. Since releases typically produce small amounts of code and the business model allows the size of the programming team to remain almost unchanged between versions, releases will appear comparatively overstaffed and are likely to exhibit lowered productivity in terms of LOC/person-month.

In order to better understand the empirical results and to further examine the interactions between the business model and a new (potentially high-productivity) technology, such as object-technology, we developed an integrated productivity-based simulation model of the software and business processes. In the following section we describe the model, and then we use it to further investigate the productivity issues.

Model

The model makes several assumptions regarding organizational capabilities and business priorities:

1. The organization is capable of meeting given deadlines. This means that the organization has in place technological capabilities for developing required software within the defined schedule and software process and risk management structures and mechanisms that are capable of ensuring with high probability that the deadlines are met.[45]
2. The projects are driven by calendar schedules, and all changes in the project requirements, personnel, or milestone dates can be represented as changes in effective team productivity over a period of time.
3. When in effect, Parkinson's Law is assumed to affect all deadlines.
4. Project deadlines are enforced only at specified milestones. The most likely deadline to be enforced is the final deadline; however, any set of deadlines can be enforced.

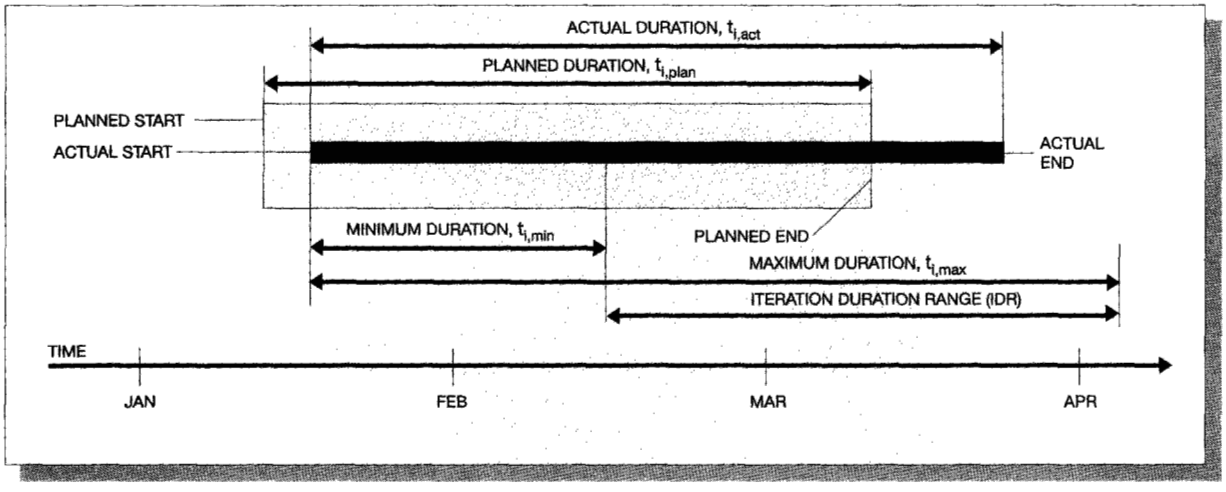
Software project iterations. The granularity of our model is at the level of project iterations, the sequences of tasks described earlier in our discussion of Table 1. In addition to an individual iteration, we recognize *aggregations of iterations*. The start of an aggregation of iterations is conditioned on completion of the iterations that precede it. Representing the duration of a project iteration as a function of team productivity requires estimation of the effective size or complexity of a project iteration (e.g., in terms of equivalent KLOC), and of the average team productivity over the iteration in the same units (e.g., in KLOC/calendar development month). The duration of an iteration is then[46]

$$\text{iteration duration (months)} = \frac{\text{iteration size (KLOC)}}{\text{team productivity (KLOC/month)}} \quad (1)$$

The relationship between iteration duration and size is linear if and only if team productivity is constant with size and time. Once an iteration completion time is determined, the duration of the overall project can be computed by adding the estimated durations of the iterations on the critical path(s) of the project, as is typically done with a PERT network.[32] For the remainder of this discussion, we will assume that we operate on the critical path of the project.

One of the characteristics of the schedules we analyzed is wide variance in the average team productivity. To incorporate this variance in the iteration duration estimates, we define the minimum and maximum team productivity and use this range to estimate the minimum and maximum iteration duration range (IDR). The minimum duration for an iteration can be achieved only if the programming team is working at their maximum productivity; this will al-

Figure 6 Planned (shaded box) and actual (heavy line) iteration durations and corresponding metrics



most certainly include code developed from overtime work. The maximum iteration duration is theoretically infinite, but in practice is usually limited by market forces, such as a fraction of an 18-month development cycle.

Figure 6 illustrates the quantitative characteristics of an iteration, i.e., the metrics we define for an individual iteration. Let the duration of an iteration be t. For each iteration i we define the actual iteration duration, which we denote by $t_{i,act}$, the planned duration $t_{i,\,plan}$, the maximum duration $t_{i,max}$, and the minimum duration $t_{i,min}$. If we assume that the size of the problem being solved in an iteration remains essentially constant during that iteration, then the minimum and maximum durations are functions of maximum and minimum team productivity, respectively, in that iteration. In practice, a new project iteration will not start before the previous iteration has completed. Hence, for each iteration we can also recognize the chain of events that lead to it, i.e., its aggregate duration, d, that includes durations of all the sequential iterations that precede it.

The aggregate duration of iteration i is a function of the sequence of iterations that precede it on the critical path, $d_i = \Sigma t_i$. For example, $d_1 = t_1$, $d_2 = t_1 + t_2$, etc. As with the individual tasks and iterations, there are four aggregate durations: actual, planned, maximum, and minimum.

The minimum and maximum duration times for an iteration define a range of possible completion times for that iteration. We represent the duration of iteration i with the random variable T_i. T_i can assume values between the minimum duration $t_{i,min}$, and the maximum duration $t_{i,max}$. Duration of a project with n successive iterations is a random variable D defined by

$$D = T_1 + T_2 + \cdots + T_n \qquad (2)$$

For example, adding the minimum iteration duration time from each iteration on the critical path, $\Sigma t_{i,\,min}$, gives the minimum duration time for the overall project, $d_{n,\,min}$. Adding the maximum task duration times, $\Sigma t_{i,\,max}$, gives the maximum duration time for the project, $d_{n,\,max}$.

To simulate the duration of a project whose iterations fall within the intervals $[t_{i,\,min}, t_{i,\,max}]$, $i = 1, \cdots, n$, we take a sample from each interval according to the distribution assigned to that interval. This provides an estimate of the individual iteration duration times for the project.

From this estimate the aggregate durations can be determined, as well as the overall project duration time. We repeat this sampling until the required simulation accuracy is achieved.

In the work reported in this paper we used a uniform distribution[47] to sample individual productivity ranges. However, note that aggregation of the iteration delays is equivalent to convolution, and that

introduction of business effects, such as the Deadline Effect and Parkinson's Law delay, requires constrained sampling of these intervals so that the resulting conditional distributions are not uniform anymore. For example, the Irwin-Hall distribution can be used to describe a general milestone distribution obtained through unconstrained convolution of uniform distributions.[36] Also, in this paper we do not address team-dependent inter-iteration correlation. Our experience is that in practice this is a lower-order effect compared to effects we deal with in the present analysis. A more detailed discussion of team-dependent correlated delays, and of the impact this has on the development process, can be found in Potok.[36]

Next we quantify Parkinson's Law and deadline enforcement, then we apply these effects to the derived iteration and project duration distributions.

Parkinson's Law. Cyril Parkinson published a collection of aphorisms about economics in 1957. Most remembered is "work expands to fill the time," or as originally stated, "work expands so as to fill the time available for its completion."[39] Gutierrez et al. have developed a stochastic model to represent the effects of Parkinson's Law on a project.[40] One of the fundamental concepts they propose is that unconstrained activity modeling (such as that seen in PERT models) may be inappropriate to represent real projects, and that completion time should be a function of the time scheduled for a project. If we consider a project iteration as a single task, then the basis for their model can be expressed as

$$d_{n,act} = \Sigma t_{i,act} + w(d_{n,\,plan}) \qquad (3)$$

where $w(d_{n,\,plan})$ is the work expansion function, and $d_{n,\,plan}$ is the project deadline. Projects under Parkinson's Law will generally not have an aggregate duration of less than the scheduled duration. That is, if $d_{1,\,plan}, d_{2,\,plan}, \cdots, d_{n,\,plan}$ are the scheduled durations for iteration aggregates, then iteration 1 is planned to complete by time $d_{1,\,plan}$, iterations 1 and 2 are planned to complete by time $d_{2,\,plan}$, and so on. We model Parkinson's Law by delaying the aggregate completion times that are less than the planned duration times. An aggregate duration that would normally be shorter than the planned deadline is expanded so that it meets the deadline, while an iteration that would normally finish after the deadline is not modified. Tasks under Parkinson's Law either finish, or are expanded to finish, within the interval $[d_{i,\,plan} - \varepsilon, d_{i,max}]$, where ε is a small time period, typically one or two weeks. The lower bound is defined by the planned aggregate iteration duration, while the upper bound is the actual maximum duration for the aggregate.

Deadline effect. According to Boehm: "The amount of energy and effort devoted to an activity is strongly accelerated as one approaches the deadline for completing the activity."[32] This effect on software is widely known but surprisingly little studied. However, in industrial psychology the effect has been thoroughly described and studied through goal theory.[43] Goal theory supports both Parkinson's Law—performance is lower if goals are easy—and the Deadline Effect—performance is higher if the deadline is challenging. The Deadline Effect depends on enforcement of milestone (task and iteration) deadlines. We model this by discarding the cases for which any of the hard deadline aggregate tasks in the case finish after their deadline. This provides us with conditional distributions for the subset of samples that meet the constraints.

For example, the combined effect of Parkinson's Law and deadline enforcement over a set of possible iteration and project durations is described by the algorithm that follows. The iteration durations are bounded by upper and lower productivity ranges and are under the influence of both Parkinson's Law and deadline enforcement. When simulating software development, we generate a number of case samples ($j = 1, \cdots, k$, e.g., $k = 100\,000$) each of which represents one complete set of project iterations ($i = 1, \cdots, n$). Function HARD() returns true if the deadline is "hard" and false if it is "soft" (i.e., allows slippage).

```
For (j = 1, ··· , k) do
    Acquire sample t1,act, ··· , tn,act for case j
    Calculate d1,act, ··· , dn,act
    Loop through all iterations (1, ··· , n)
      If di,act < (di,plan − ε) then
        expand iteration duration to the deadline
        recompute current and all remaining di,act
                                          (i, ··· , n)
      EndIf
    EndLoop
    Loop through all di,plan (1, ··· , n)
      If [HARD (di,plan) and di,act > di,plan] then
        discard this case and exit loop
      EndIf
    EndLoop
EndFor
```

This algorithm, which is equivalent to constrained discrete convolution of iteration completion times,[36] provides a model for how constrained iterations complete around a given milestone.

Maturity. It is worth noting that the same simulation models allow us to examine the influence of the organizational software process maturity, and of the maturity of software development technology, by varying the iteration (task) productivity ranges (or IDRs). For example, we would expect that mature software processes and technology would promote small productivity variance (i.e., smaller duration range in Figure 6), while poor process control or immature technology may result in a much wider range of productivities and consequently larger iteration (task) duration ranges.

Simulation

We first use the simulation model to explore the four business models and their effect on productivity. We then model and discuss the effect of process maturity.

Projects. In the simulations presented here we use two hypothetical projects. One project has a "normal" team productivity range in each iteration. The second project differs from the first only in that the upper bound on its team productivity range is twice as large. Both sets of ranges have the same lower bound. In the examples given below, we assume that the development *team productivity*[48] for the first project ranges from 500 LOC/week to 1250 LOC/week, while the range for the second project team is from 500 LOC/week to 2500 LOC/week. From Equation (1) it follows that a project of the normal type has iteration duration range from 8 to 20 weeks, while for a project of the second type the IDR is 4 to 20 weeks. Iteration duration times are sampled from these ranges assuming a uniform distribution. Note that the productivity difference between the two types is consistent with empirical studies of differences in productivity between noncommercial procedural and object-oriented projects.[3,5]

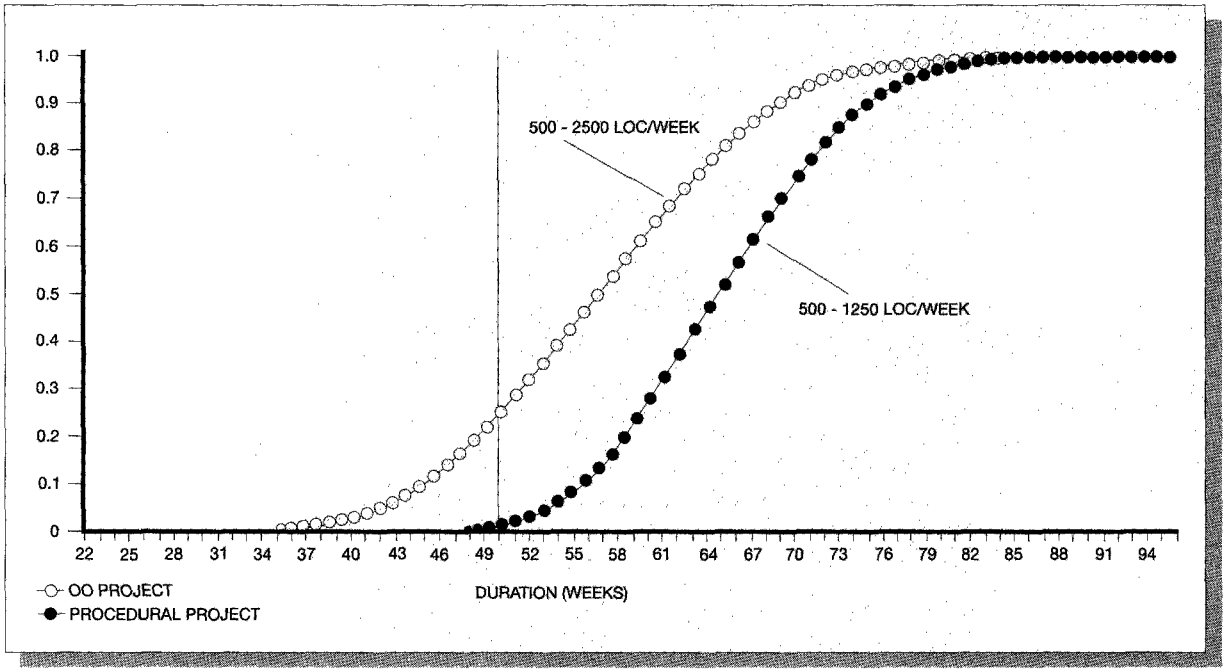
The normal range could be considered as procedural development, and the second range could represent a development technology that has potential for greater average productivity, such as object technology. We will sometimes refer to the high average productivity project as the "object-oriented project" and to the other one as the "procedural project." Note that since the object-oriented project has the same lower productivity bound as the procedural project but a higher upper bound, the sample set from the object-oriented project IDR has higher variance. This suggests that there is less control over the development process that implements the new technology. Of course, this does not have to be the case, and a new technology could offer not only a higher mean value for its IDR (as compared with procedural IDR), but also a smaller IDR range around that mean value. We discuss this later.

We also make the following assumptions. When a business model is applied, both projects start at the same time and operate under the same milestone constraints. When Parkinson's Law is in effect the IDR lower bound is no earlier than one week before the deadline. When deadline enforcement is active on iteration i, its IDR upper bound is the same as the planned deadline ($d_{i,\,plan}$). These restrictions are consistent with our empirical data. Both simulated projects are assumed to have five equally sized iterations. The planned duration for each iteration is set to 10 weeks. This translates to planned deadlines at 10, 20, 30, 40, and 50 weeks, respectively. We also assume that an equivalent of 10 KLOC is developed during each iteration, i.e., a total of 50 KLOC per project.

We would expect that mature processes and technology would promote small productivity variance.

Business models. Our first case, Model A, simulates the effect of a business practice that provides incentive to finish a project as soon as possible, with no penalty for finishing late. This represents the situation where the process can be viewed as being free from both deadline and Parkinson's Law effects. The cumulative distributions for the duration of the projects under this model are shown in Figure 7. The mean completion time for the object-oriented project is about 58.8 weeks, with a standard deviation of 10.0 weeks, while the mean for the procedural project is about 67.5 weeks, with standard deviation of 7.8 weeks. Further, only about 20 percent of the ob-

Figure 7 Cumulative distributions for high and low productivity projects without milestone constraints (Business Model A)



ject-oriented samples finish at or before the Week 50 deadline, and only about 1 percent of the procedural samples complete in this time frame. It is obvious that, for the type of project in question, the 50-week deadline is quite aggressive and exceeds the capability of either technology to reliably meet it. However, an object-oriented approach still has a better chance to do so than the procedural project. On the other hand, if we assume that the final project deadline is 72 weeks, we see that the higher-productivity methodology has over 90 percent chance of meeting it. For comparison, the COCOMO (Constructive Cost Model)[32] average for a 50-KLOC project is between 145 and 240 person-months, with a completion time of 17 months (or about 68 to 70 weeks).

The second case, Model B, provides no incentive for finishing early and no penalty for finishing late. This represents the situation where Parkinson's Law is in effect for all milestones. Figure 8 shows the resulting project completion distribution. The mean completion time for the object-oriented project is now 61.3 weeks, two and a half weeks longer than under Model A, but with a smaller standard deviation of eight weeks. On the other hand, the procedural project results differ only slightly from the Model A case, with a mean of 67.8 weeks and standard deviation of 7.5 weeks. Since under Model B assumptions there is no incentive to finish early, we would expect that any iterations that naturally complete early would be prolonged. This reduces the fraction of projects that finish before or on the 50-week deadline by nearly 10 percent for object-oriented samples, but by only about 0.5 percent for the procedural samples. It is obvious that this lack of incentive to complete early, i.e., Parkinson's Law delay, has greater impact in the case of the object-oriented project. Deadlines are often set so that the product can be favorably marketed; for example, product releases may be timed for shipment with a new version of an operating system. Delays can limit or negate the potential productivity gain a technology may offer.

Under this business model, the average project duration and variance become more similar for the two methodologies and the potential for productivity gains from object-oriented development is less pronounced. This is illustrated further in Figure 9, which shows the corresponding estimated probability density functions. We see that the delay in the early com-

Figure 8 Cumulative distributions for high and low productivity projects conducted under Parkinson's Law (Business Model B)

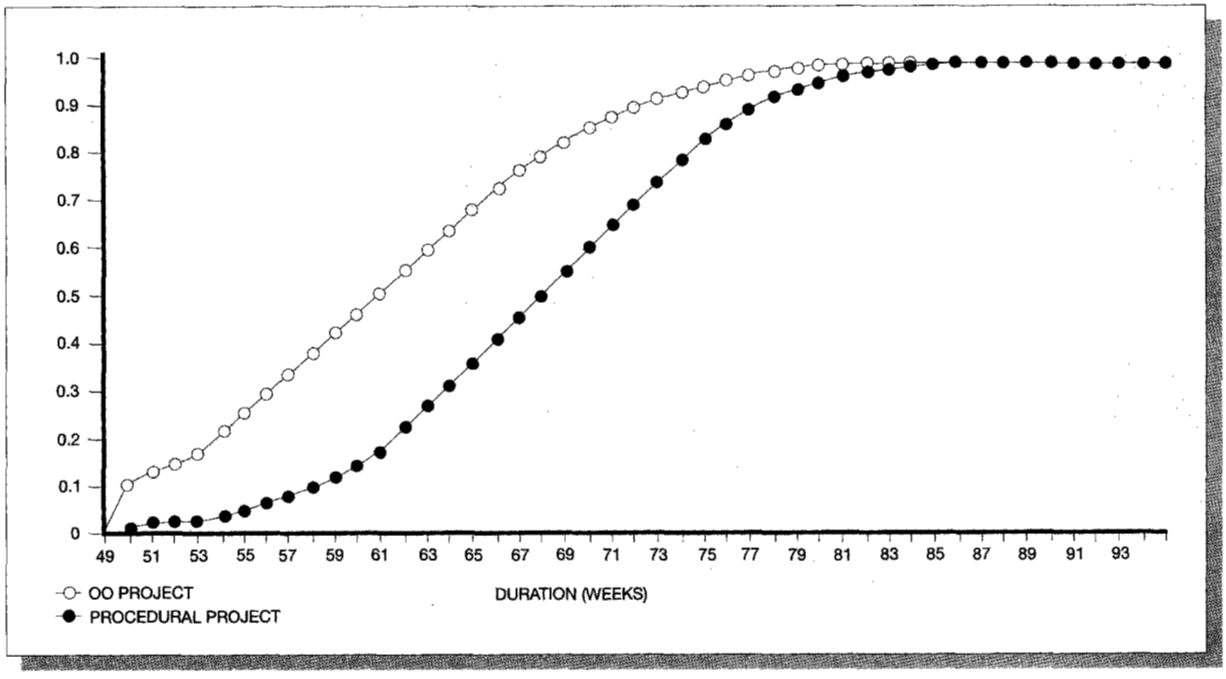


Figure 9 Probability density functions for high and normal productivity projects with Parkinson's Law applied (Business Model B)

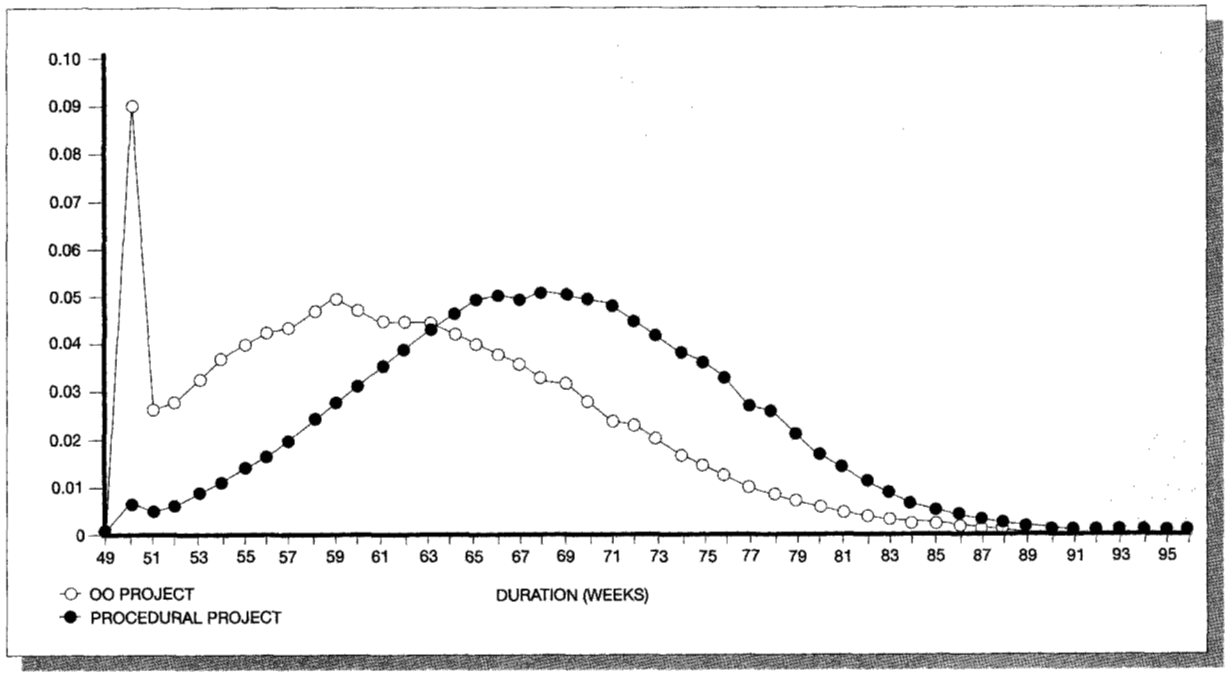
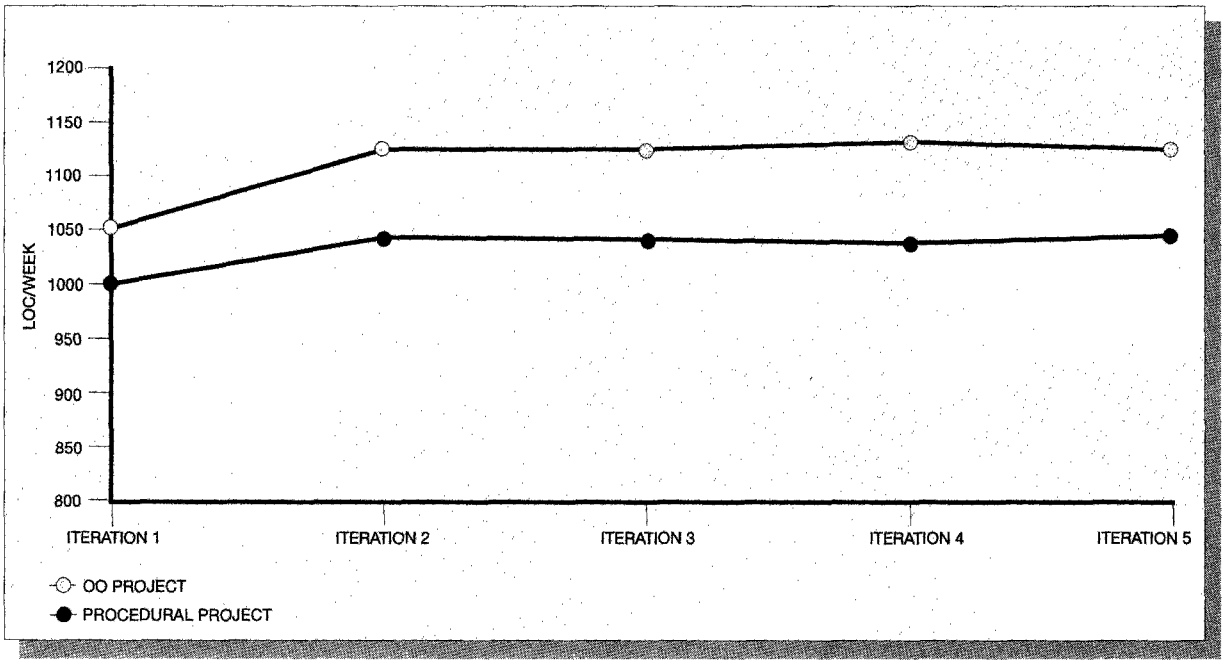


Figure 10 Average team productivity required to complete the project by Week 50 deadline (Business Model C)



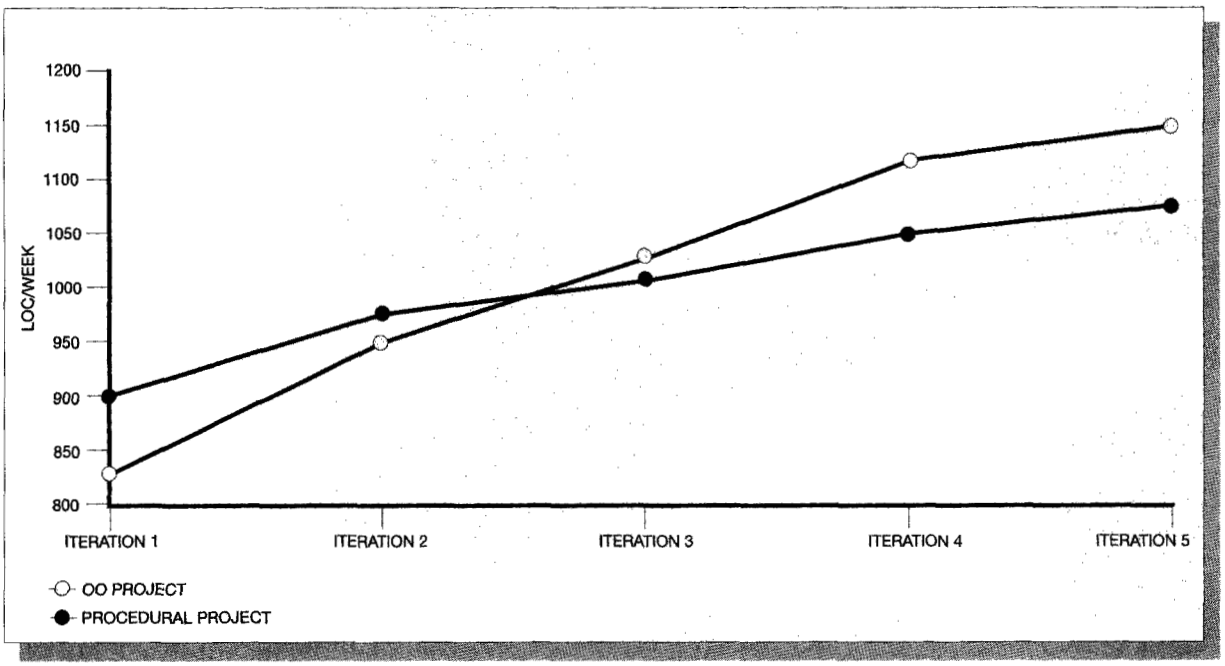
pletions translates into a relaxation spike at Week 50, but we also see that this effect is more pronounced in the case of the potentially more productive methodology.

Our third case, business Model C, provides incentive for finishing early and a penalty for finishing late. This is conceptually the same as enforcing the final deadline but with no Parkinson's Law effect. As mentioned earlier, only about 20 percent of the object-oriented samples and only 1 percent of procedural samples make the deadline. The mean duration for the 20 percent of the object-oriented projects that naturally make the deadline is 44.6 weeks with standard deviation of 4.3 weeks, while the average duration for the 1 percent of the procedural projects that naturally make the deadline is 47.9 weeks with standard deviation of 1.7 weeks. The average team productivity computed for these samples at each of the five iterations is shown in Figure 10. We see that naturally successful projects have some "slack" in the first iteration, but after that software personnel must maintain an average productivity that is roughly in the middle of their range if object-oriented development is used, and about 80 percent of their maximum for procedural development. It is very likely that the latter requirement will put more strain on the software team, since 80 percent of maximum productivity probably implies overtime work.

In practice, it is unlikely that either of the three model situations will occur in pure form. For example, it is unrealistic to assume no penalty for late completion, and it is probably equally unrealistic t › assume that incentives to finish early are 100 percent effective. A great deal of planning and effort is required to ship a product, and changing the ship date late in the cycle is costly whatever the reason and direction. Based on the workflows analyzed in this study, Model D is a more realistic business situation, with little or no incentive to finish earlier than planned and a penalty for finishing late. Conceptually, this is the same as adding both Parkinson's Law and deadline enforcement to a project.

In Figure 11 we show the average team productivity needed around the five modeled milestones, assuming that all five operate under Parkinson's Law and that only the final project deadline ($d_{5,plan}$) is enforced. We see that for projects that meet the final

Figure 11 Team productivity per task for both projects under Business Model D



deadline, the average productivity increases steadily as we approach it. Of course, projects that do not naturally conform to these curves require explicit manipulation of their productivity in order to meet the final deadline and that gives rise to the different productivity envelopes and slopes that the Deadline Effect generates.[32] There is another interesting feature that we see in Figure 11. A successful object-oriented project allows the teams more slack (lower productivity) in the early project stages, which means that they can operate in a more relaxed fashion than procedural teams. This results in early deadlines being missed, under the expectation that the final deadline is not in jeopardy.

Another interesting view of Model D is provided in Figure 12. It shows estimated iteration-duration probability-density distributions around the intermediate deadlines for all project samples that meet the final deadline. The planned duration for each iteration is 10 weeks, so the deadline for Iteration 1 is 10 weeks, for Iteration 2 is 20 weeks, and so on, with the planned finish for Iteration 5 at 50 weeks. At the start of each iteration we see the Parkinson's Law relaxation spike. The shape of the curves shown in Figure 12 is similar to the shapes observed empirically in Figures 3 through 5.

From Figure 12 we see that while both the procedural and the object-oriented projects finish at the same time, and thus have the same overall average productivity with respect to calendar time, the business process appears to cause a reduction in variance around the deadlines as the projects progress. This results in a reduction in the distribution "tails" across successive iterations. That is, the hard deadline at the end of Iteration 5 forces earlier completions in the iterations closer to the final deadline, and in this way acts as a variance-reduction mechanism. This is interesting, since in an unconstrained development process, convolution of iteration completion times would result in increasing, rather than decreasing duration variance.[49] Furthermore, the lower productivity project (filled circles) requires this better process control (i.e., shorter distribution tails that imply lowered variance around the intermediate deadlines) in order to meet the final deadline, while the higher productivity project (hollow circles) can have longer tails (it can slip many of the inter-

Figure 12 The marginal distribution of successful projects around the individual task deadlines under both Parkinson's Law and the Deadline Effect (Business Model D)

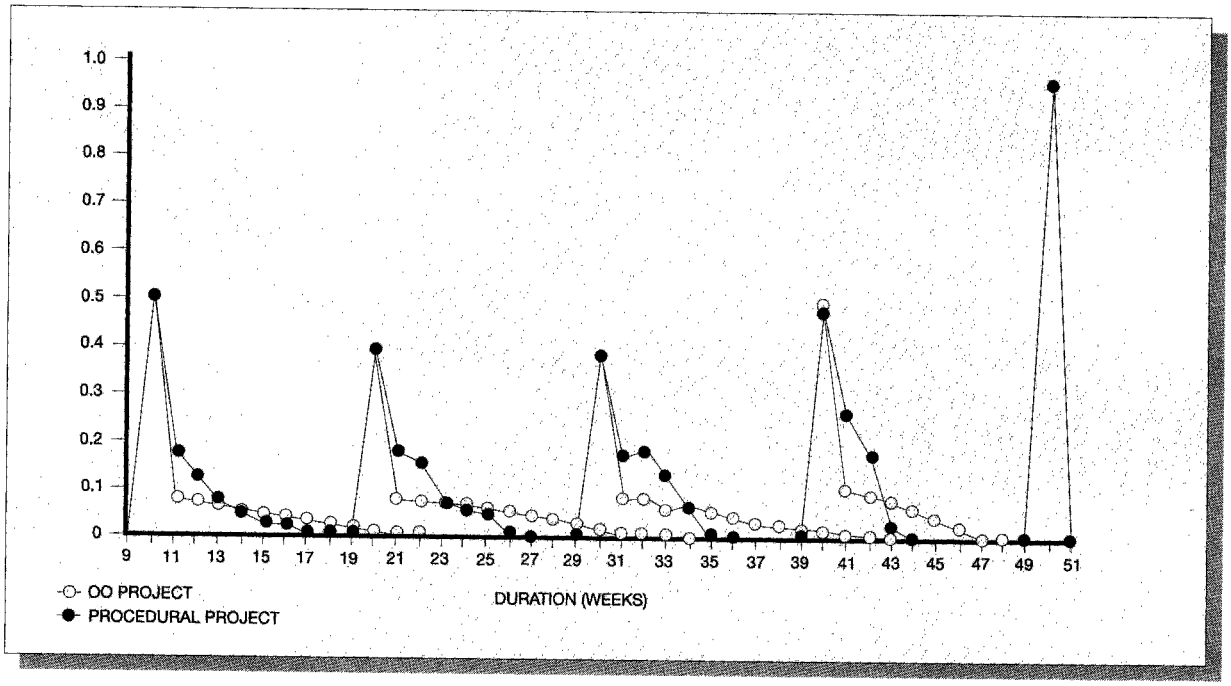


Figure 13 Comparison of duration for two projects with differing productivity capabilities

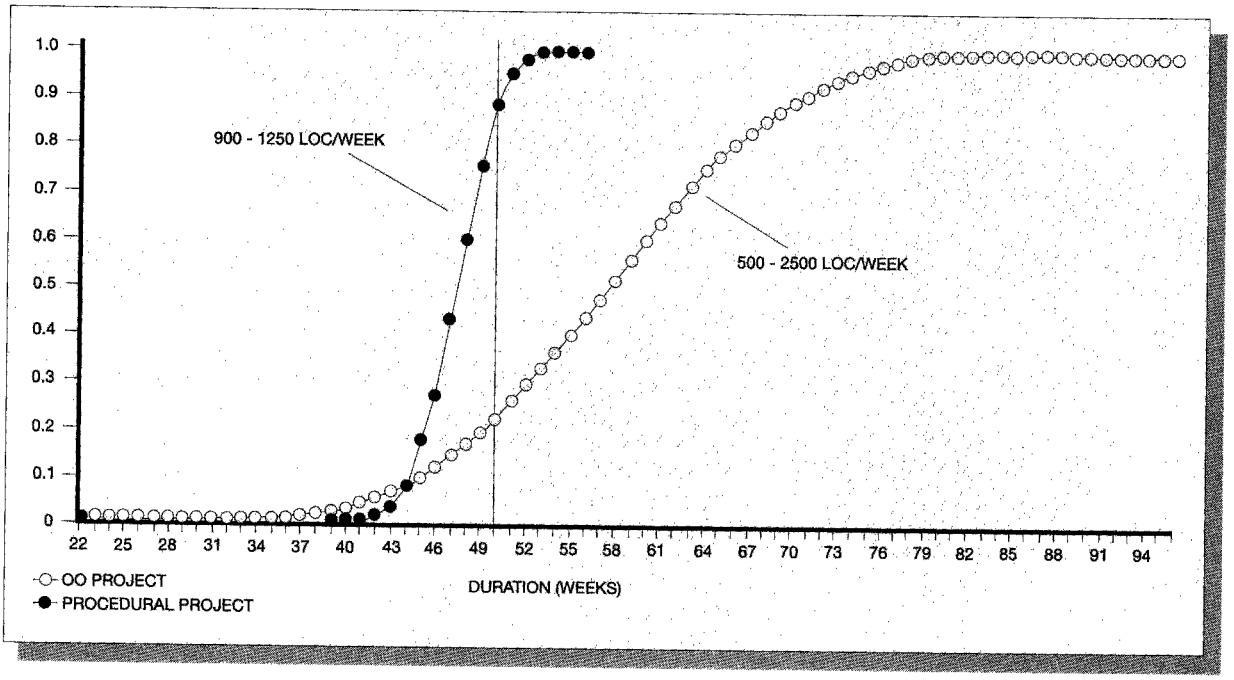
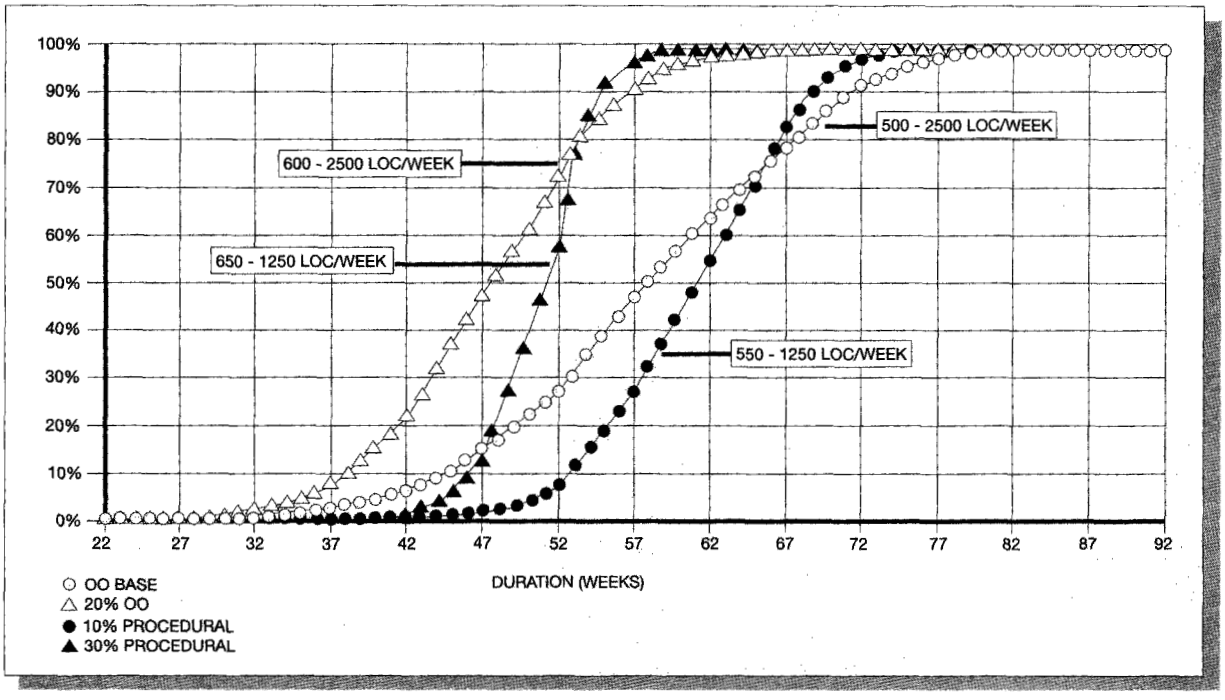


Figure 14 Influence of the mean and width of the productivity range



mediate milestones) and still meet the final deadline.

Maturity and process control. So far we have assumed that the two modeled projects have the same lower bound on iteration productivity. This means that the worst-case scenario in both methodologies produces projects of the same duration. It could be argued that a mature technology may have a productivity range with less variance than a new technology, even if the new technology has potential for higher productivity. Similarly, an organization may have better control over projects that use a classical methodology than over those that use a new technology. This may translate into a narrower iteration duration range when using better controlled, mature processes. We illustrate the effect of narrower IDRs in Figure 13.

In Figure 13 we show the unconstrained cumulative distribution (Business Model A) for projects that have average team productivity of 1075 LOC/week (filled circles) and 1500 LOC/week (hollow circles), respectively. However, the lower average productivity project has a productivity range between 900 LOC/week and 1250 LOC/week, while the higher average productivity project range is between 500 LOC/week and 2500 LOC/week. Comparison with Figure 7 shows a striking difference. With the increased control over the process (reduced IDRs), 90 percent of the procedural projects are now capable of making the 50-week deadline. This is a far higher percentage than is seen with the object-oriented approach, even though the latter has potential for twice the productivity of the procedural project. In examining real products we have encountered procedural projects that did better than object-oriented projects and the reverse. Inspection of Figures 3 through 5 shows that the normalized task-completion delay distributions are less spread out for the object-oriented projects than for the procedural project.

It is obvious that control over the width of the productivity range (and by implication, control of the software process) can play an even more important role than potential productivity gains from a new technology. While business-imposed incentives and deadlines can have an important impact on the perceived team productivity and on the probability that a project finishes on time, it may be even more im-

portant to select the methodology and the development approach over which an organization has good control with acceptable productivity and as narrow an IDR as possible. Doing so may increase the probability of project completion by a given deadline, even when the deadline is aggressively set by market forces.

In Figure 14 we illustrate possible trade-off issues between better process control (as manifested by the width of the productivity range) and productivity potential (as manifested by the average productivity and the upper productivity bound). Figure 14 shows cumulative project completion distributions for four cases: (1) the basic object-oriented project (range: 500–2500 LOC/week/team, average team productivity of 1500 LOC/week), (2) a project with the same upper bound on team productivity but a 20 percent better lower bound (range: 600–2500 LOC/week, average: 1550 LOC/week), (3) a procedural project with average team productivity of 900 LOC/week (range: 550–1250 LOC/week), and (4) a project where the average team productivity is 950 LOC/week and the range is 650–1250 LOC/week. Inspection of the figure and comparison with Figure 7 shows a number of interesting things.

Both Parkinson's Law delays and the Deadline Effect tend to be the result of the applied business model.

For example, from Figure 7 we see that the probability that a project with a procedural productivity profile (500–1250 LOC/week) completes within 67 weeks is about 50 percent, while the probability that a project with our assumed basic object-oriented profile (500–2500 LOC/week) completes within 67 weeks is about 80 percent. However, Figure 14 shows that if we increase the low-end productivity of the procedural profile by only 10 percent we can raise its completion probability to 80 percent. That is, a 10 percent increase in the lower productivity bound (from 500 to 550 LOC/week) has the same effect as doubling the upper bound. Similarly, the probability that the basic object-oriented project completes by Week 55 is about 36 percent. To achieve 85 percent completion probability around Week 55 we could choose a 20 percent improvement in the lower bound of the basic object-oriented team productivity range or a 30 percent improvement in the lower bound of the procedural profile. However, we have to remember that business effects can counteract these improvements. For instance, comparison of basic object-oriented profiles in Figures 14 and 8 shows that lack of incentives to complete a project before Week 50 reduces the probability of project completion at Week 55 from about 0.36 to about 0.25.

It is obvious that an organization has a number of options for improving the probability that its projects complete by some deadline. Improvements in the process that result in a small increase in the lower productivity bound (e.g., improved training of the personnel in the use of the technology) can be as effective as a shift to a new technology that has considerably higher productivity potential but may be implemented with less control (i.e., with a broader productivity range). Of course, other effects, such as those of the business model, also have to be taken into account.

Conclusions

As market pressures shorten software development cycles, an increasing emphasis is being placed on improving software development productivity. Object-oriented software development has emerged as a potential solution, i.e., as technology with great potential for reducing product time to market. While this may be true in cases where high levels of design and code reuse are present (which can be achieved without object technology as well), there is little evidence that this occurs in the first few product generations, at least not for commercial projects operating under a common business model.

In this paper we reported on empirical and simulation-based studies of the relationship between common commercial business practices and the software productivity that might be expected in such an environment. Our data indicate that object-oriented projects suffer from Parkinson's Law delays, and from the Deadline Effect, in much the same way that procedural projects do. Both effects tend to be the result of the applied business model. For example, a rigorous enforcement of final project deadlines, coupled with a lack of incentive to finish intermediate project tasks early, may trigger Parkinson's Law delays and negatively influence productivity. This ef-

fect may become especially evident when projects are developed using potentially higher productivity methods, such as object technology, but operate under business models and deadlines that are more suited for productivity expected from classical methodologies.

We used simulation to show that while a methodology with potential for higher productivity may enable software development teams to operate in a less stressful mode, the promise of high productivity alone is not enough. An organization must be able to control the range of productivities in which its development teams operate. A wider range implies less control over the process and less ability to guarantee timely project completion. The decision to use a new technology should be based not only on its promised maximum, or even average, productivity but also on the ability of the organization. If the business model cannot adjust to new technology by recognizing its limitations, assessing the ability of the organization to control it, and adjusting deadlines to take advantage of its potential, it is unlikely that an investment in the technology will result in real productivity benefits.

Acknowledgments

We would like to thank Dan Blum, Chris Wicher, and the IBM Software Solutions Laboratory in Research Triangle Park (RTP) for their strong support of this research, and Paritosh Dixit of North Carolina State University (NCSU) for his assistance with the statistical analyses. We are grateful to the anonymous referees for their constructive comments that have helped to improve the organization and presentation of this work. Work was supported in part by IBM Canada (Centre for Advanced Studies, Toronto), by IBM RTP, by the IBM Shared University Research program, and by NCSU Center for Advanced Computing and Communications.

Cited references and notes

1. The latest newcomer in this arena is the Java language and Web-based software development.
2. I. Jacobson, M. Ericsson, and A. Jacobson, *The Object Advantage: Business Process Reengineering with Object Technology*, Addison-Wesley Publishing Co., Wokingham, UK (1994).
3. J. A. Lewis, S. M. Henry, and D. G. Kafura, "An Empirical Study of the Object-Oriented Paradigm and Software Reuse," *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, Phoenix, AZ, October 6–11, 1991, pp. 184–196.
4. In this experiment, the reuse level may have been as high as 25 percent in some cases.
5. W. L. Melo, L. C. Briand, and V. R. Basili, *Measuring the Impact of Reuse on Quality and Productivity in Object-Oriented Systems*, Technical Report CS-TR-3395, Department of Computer Science, University of Maryland, College Park, MD (January 1995).
6. For high-end productivity gains reuse levels were in the range of 40–50 percent.
7. B. Henderson-Sellers, "The Economics of Reusing Library Classes," *Journal of Object-Oriented Programming* **6**, No. 4, 43–50 (1993).
8. D. Schimsky, "Software Reuse: Some Realities," *Vitro Technical Journal* **10**, No. 1, 47–57 (1992).
9. D. A. Boehm-Davis and L. S. Ross, "Program Design Methodologies and the Software Development Process," *International Journal of Man Machine Studies* **36**, No. 1, 1–19 (1992).
10. H. Zweben, S. H. Edwards, B. W. Weide, and J. E. Hollingsworth, "The Effects of Layering and Encapsulation on Software Development Cost and Quality," *IEEE Transactions on Software Engineering* **21**, No. 3, 200–208 (1995).
11. G. Booch, *Object-Oriented Design with Applications*, The Benjamin/Cummings Publishing Co., Redwood City, CA (1991).
12. D. deChampeaux, D. Lea, and P. Fauve, *Object-Oriented System Development*, Addison-Wesley Publishing Co., Reading, MA (1993).
13. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, Inc., Englewood Cliffs, NJ (1991).
14. R. Wirfs-Brock, B. Wilkerson, and L. Wiener, *Designing Object-Oriented Software*, Prentice Hall Inc., Englewood Cliffs, NJ (1990).
15. D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes, *Object-Oriented Development: The Fusion Method*, Prentice Hall, Inc., Englewood Cliffs, NJ (1994).
16. F. Hayes and D. Coleman, "Coherent Models for Object-Oriented Analysis," *Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications*, Phoenix, AZ, October 6–11, 1991, pp. 171–183.
17. D. E. Monarchi and G. I. Puhr, "A Research Typology for Object-Oriented Analysis and Design," *Communications of the ACM* **35**, No. 9, 35–47 (1992).
18. B. Henderson-Sellers and J. M. Edwards, *BOOK TWO of Object-Oriented Knowledge: The Working Object*, Prentice Hall, Inc., Sydney, Australia (1994).
19. L. Berlin, "When Objects Collide: Experiences with Reusing Multiple Class Hierarchies," *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, Ottawa, Canada, October 21–25, 1990, pp. 181–193.
20. M. F. Dunn and J. C. Knight, "Software Reuse in an Industrial Setting: A Case Study," *Thirteenth International Conference on Software Engineering*, Austin, TX, May 13–16, 1991, pp. 381–390.
21. E. H. Gamma, R. Johnson, and J. Vlissides, "Design Patterns: Abstraction and Reuse of Object Oriented Design," *Proceedings of the Seventh European Conference on Object-Oriented Programming* (1993).
22. M. L. Griss, S. S. Adams, B. Howard, B. J. Cox, and A. Goldberg, "The Economics of Software Reuse (Panel)," *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, Phoenix, AZ, October 6–11, 1991, pp. 264–270.
23. W. Wessale, D. Reifer, and D. Weller, "Large Project Ex-

periences with Object Oriented Methods and Reuse," *Journal of Systems Software* **23**, No. 2, 151–161 (November, 1993).
24. G. A. Hansen, "Simulating Software Development Processes," *IEEE Computer* **29**, No. 1, 73–77 (1996).
25. T. E. Potok and M. A. Vouk, "Development Productivity for Commercial Software Using Object-Oriented Methods," *Proceedings of the 1995 CASCON Conference*, Toronto, Canada, October, 1995.
26. T. E. Potok and M. A. Vouk, *Productivity of Object-Oriented Software Development*, Technical Report CACC-TR-96/31, Center for Advanced Computing Communications, North Carolina State University, Raleigh, NC (1996).
27. Data used by permission. The scales appearing on the axes of all graphs, and any product and date-related information, have been altered to provide discretion.
28. Procedural software development uses structured analysis, design and coding, or similar techniques, to develop and implement a software product.
29. ISO 9000 is a set of specifications and standards for quality assurance management systems. It was written by representatives from the 91 countries that are members of the International Organization for Standardization (ISO). ISO 9000 certification is granted after successfully passing an external audit against the ISO 9001 Standard, made up of 20 elements that define acceptable quality management systems.
30. The Malcolm Baldrige National Quality Award recognizes United States companies for business excellence and quality achievement. The criteria are focused on customer satisfaction, continuous improvement, and business results. To be considered for this annual award, a company submits an application that describes its efforts against the criteria. Business and quality experts review the applications to determine the winner.
31. However, the advent of the World Wide Web has introduced a new category of Web-based applications that require development and release cycles of around three calendar months (one "web year"). This opens some interesting process control issues, some of which we discuss in the paper.
32. B. W. Boehm, *Software Engineering Economics*, Prentice Hall, Inc., Englewood Cliffs, NJ (1981).
33. R. E. Fairley, *Software Engineering Concepts*, McGraw-Hill, Inc., New York (1985).
34. It is interesting to note that a "web-year" cycle could be cast as an "iteration" with a very hard iteration deadline.
35. Note that in "web-year" type development full testing would be enforced at the end of each web-year iteration.
36. T. E. Potok, *Development of a Quantitative Process Model for Object-Oriented Software Development*, doctoral thesis, Department of Computer Science, North Carolina State University, Raleigh, NC (1996).
37. M. A. Vouk, "On the Cost of Mixed Language Programming," *ACM SIGPLAN Notices* **19**, No. 12, 54–60 (1984).
38. Based on interviews with the workflow owners and on a review of the project documentation.
39. C. N. Parkinson, *Parkinson's Law and Other Studies in Administration*, Houghton Mifflin Company, Boston, MA (1957).
40. G. J. Gutierrez and P. Kouvelis, "Parkinson's Law and Its Implications for Project Management," *Management Science* **37**, No. 8, 990–1001 (August 1991).
41. D. S. Borger and M. A. Vouk, "Modeling the Behaviour of Large Software Projects," *Center for Communications and Signal Processing*, Technical Report TR-91/19, North Carolina State University, Raleigh, NC (1991).
42. As of 1990, over 400 experiments have been performed testing this theory, with over 90 percent supporting it.
43. E. A. Locke and G. P. Latham, *A Theory of Goal Setting and Task Performance*, Prentice Hall, Inc., Englewood Cliffs, NJ (1990).
44. G. P. Latham and H. A. Marshall, "The Effects of Self-Set, Participatively Set and Assigned Goals on the Performance of Government Employees," *Personnel Psychology* **35**, 399–404 (1982).
45. M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber, "Capability Maturity Model, Version 1.1," *IEEE Software* **10**, No. 4, 18–27 (July 1993).
46. A. B. Badiru and P. S. Pulat, *Comprehensive Project Management: Integrating Optimization Models, Management Principles, and Computers*, Prentice Hall, Inc., Englewood Cliffs, NJ (1995).
47. One could argue that a triangular or beta distribution may be a good starting representation for the iteration duration. However, there is no evidence that independent unconstrained iteration durations have a specific distribution, so we have opted for the more general uniform distribution.
48. It is also assumed that the average team size is about ten software professionals.
49. S. E. Elmaghraby, E. I. Baxter, and M. A. Vouk, "An Approach to the Modeling and Analysis of Software Production Process," *International Transactions in Operational Research* **2**, No. 1, 117–135 (1995).

Accepted for publication September 11, 1996.

Thomas E. Potok *IBM Software Solutions Division, P.O. Box 12195, Research Triangle Park, North Carolina 27709 (electronic mail: potok@vnet.ibm.com).* Dr. Potok is an advisory programmer at the Software Solutions Laboratory of IBM in Research Triangle Park, North Carolina. He is currently merging the Application Development and Database for the Software Solutions RTP lab. He has successfully led the lab in achieving ISO 9000 certification. Prior to this, he led a team in creating an object-oriented data model designed to work with CASE (computer-assisted software engineering) tools to improve application development and quality. He has led and been a member of various other software development efforts. He has a B.S. degree in computer science, an M.S. degree in computer engineering, and a Ph.D. degree in computer engineering, all from North Carolina State University. He has authored 11 publications and has filed 2 patents.

Mladen A. Vouk *Department of Computer Science, North Carolina State University, Box 8206, Raleigh, North Carolina 27695 (electronic mail: vouk@adm.csc.ncsu.edu).* Dr. Vouk received B.Sc. and Ph.D. degrees from the University of London, United Kingdom. He has extensive experience in both commercial software production and academic computing environments. He is the author or coauthor of over 100 publications. He is currently a professor of computer science at North Carolina State University. His research and development interests include: software process and risk management, software testing and reliability, scientific problem-solving work flows, advanced high-performance networking, coding theory, and computer-based education. He teaches courses on software engineering, software testing and reliability, software process and risk management, and communication networks. Dr. Vouk is chairman of the IFIP (Internation-

al Federation for Information Processing), Working Group 2.5 on Numerical Software. He is also a senior member of IEEE (Institute of Electrical and Electronics Engineers) and a member of the Reliability Society, Communications Society, IEEE Computer Society, IEEE Technical Committee on Software Engineering, ACM (Association for Computing Machinery), American Society for Quality Control, and Sigma Xi. He is an associate editor of *IEEE Transactions on Reliability*.

Reprint Order No. G321-5639.