Simulation

Simulation: Transactions of the Society of Modeling and Simulation International
87(11) 947–975

DOI: 10.1177/0037549710382423
sim.sagepub.com

The system dynamics of Brooks' Law in team production

David G Chernoguz

Abstract

When a project is behind schedule, managers usually respond by bringing in additional workforce. Brooks' Law points out the systemic limits to staffing intervention, which are related to dynamic relationships between team buildup and loss of production due to communication and training overheads. ProjScout™, a theoretical model of Brooks' Law, responds logically when input parameters are assigned extreme values and coherently sensitive to stepwise changes of input. Simulation experiments revealed relative inherent limits of staffing intervention, related to dynamic relationships between initial team size, training new team members, their learning curve, and productive time lost due to intercommunication. As a rule of thumb, Brooks' Law applies only when project resources are limited. The actual effect of investment in training, as well as concluding whether Brooks' Law applies in a particular case, can be modeled only when actual values of the productivity, communication entropy, and actual learning curve are known. ProjScout™ can facilitate project decision-making in a wide range of production and service industries – wherever experienced performers have to divide their work time between production and other activities.

Keywords

bazaar and cathedral development, Brooks' Law, communication overhead, programming teams, schedule estimation, simulation, software development, system dynamics, training

> God grant me the serenity to accept the things I cannot change; courage to change the things I can; and wisdom to know the difference (Reinhold Niebuhr).

1. Introduction

Late product delivery and budget overruns are not limited only to software development – documented cases span from the Bible to general acquisition practice.[1–3] According to the Standish Group, perhaps as many as 90% of all software projects fail to deliver in terms of budget, schedule or functionality.[4] Frederick P. Brooks, Jr, once compared software projects to a werewolf: ' innocent and straightforward, but capable of becoming a monster of missed schedules, blown budgets, and flawed products'.[5]

At the start of a project, managers estimate the workforce required, negotiate a schedule and budget and, just in case things do not go as planned (and they almost never do), plan feasible corrective actions. Never expecting serenity in a turbulent business environment, managers want to know the difference between the things they can change and those they cannot. For instance, having people work overtime or adding manpower is often accompanied by unintended negative consequences.[6] Working overtime starts a vicious cycle, where employee burnout soon causes productivity decay.[7,8] The paradoxical effect of added manpower, or 'Brooks' Law', is regarded often as a management gospel[9] or even a truism.[10] In his seminal book, *The Mythical Man-Month* (TMMM), first published in 1975, Brooks[11] warns managers against blindly making the instinctive fix to a late project:

Department of Neurology, Hebrew University-Hadassah Medical Center, Jerusalem, Israel.

Corresponding author:
Dr David G. Chernoguz, Department of Neurology, Hebrew University-Hadassah Medical Center, P.O. Box 12000 Jerusalem 91120 Israel
Email: chassida@yahoo.com

'When schedule slippage is recognized, the natural (and traditional) response is to add manpower Like dousing a fire with gasoline, this makes matters worse, much worse. Oversimplifying outrageously, we state Brooks' Law: Adding manpower to a late software project makes it later'.

Brooks explains his law by the very nature of programming work, which is 'more like having a baby than picking cotton': 'Ten people can pick cotton ten times as fast as one person because the work is almost perfectly partitionable, requiring little communication or coordination. But nine women can't have a baby any faster than one woman can because the work is not partitionable'.[11] Unlike manufacturing, software construction is an inherently systemic effort: it cannot be easily partitioned into isolated, independent tasks. The complexity of software development ' creates the tremendous learning and understanding burden that makes personnel turnover a disaster'.[5]

An essential part of the project's information exists in a tacit form that can be conveyed only personally. When new people are brought into a project, they need to be introduced to project domain and architecture, organization policies and procedures, team responsibilities, and more. Usually, a project veteran helps a newcomer to become a part of the team. Mentoring duty entails distraction from ongoing work and increases overall communication overhead. Gerald Weinberg[12] points out that the increase in training and communication coordination overhead contributed the most to Brooks' Law, since both factors manifest themselves as added work. The training load on the experienced workers leads to a reduced amount of productive work being done.[13] As a rule of thumb, Brooks' Law predicts that while the complexity and communication costs of a project rise with the square of the number of developers, work done only rises linearly.[9] On balance, more effort is lost to training, coordination, and communication than is gained when the new staff eventually becomes productive.[9]

The prediction of Brooks' theory is supported by numerous empirical data.[14–16] The research repeated in 1996 and in 2005 using a database of 7,200 software projects confirms that adding personnel yields a declining, marginal benefit to schedule compression and an increasing negative impact on software quality.[15] The analysis by Verner et al.[17] shows that project failures are indeed largely attributable to problems discussed in TMMM.

Nevertheless, adding people to a late software project remains a commonplace practice.[9] A 1995 study indicates that nearly half of the software project managers exceeding their planned schedules or budgets by more than 30% attempted to control projects by adding staff.[18] A longitudinal citation context analysis of Brooks' Law by McCain and Salvucci,[19] appeared in 2006, shows that little has changed since Brooks first published his law. 'It is astounding that there are software engineering managers who are either unaware of the Mythical Man-Month theory, or who are, but believe it doesn't apply to their project,' writes John Gruber, referring to the 'Mythical Man-Month' disaster at Apple, when the Aperture product team grew from 20 to 150 people in just a few weeks, 'and not surprisingly, this is precisely when things went from bad to worse with regard to the quality of the product'.[20]

So why do managers believe that Brooks' Law does not apply to their case? If we do not know whether the law applies, should we pretend that it does not exist?

The rejection can be explained probably by a debilitating effect of Brooks' Law-awareness on managers' ability to deal with schedule problems, since the only option left would be client management, by negotiating schedule extension or decreasing project size. However, the estimated schedule in turn affects what actually takes place in a project. If the schedule is underestimated, planning inefficiencies are introduced, invariably leading to schedule delays.[21] If the schedule is overestimated, Parkinson's Law goes into effect: work expands to fill the time available for its completion. Allowing for extra time may also endanger the project by adding unexpected functions and unnecessary gold plating, thus potentially leading to increased schedules.[21]

1.1. Brooks' Law debate

Since its discovery, Brooks' Law set off an emotional debate in the management literature. Questioning if it does in fact contribute to the project's schedule problems, Steve McConnell turns to the poor accuracy of project estimation and tracking:[9] 'The [Brooks'] claim is that adding staff to a late project makes it later–but later than what? Later than a systematic, well-founded estimate, or later than an estimate that was optimistic by more than 100% in the first place?' Also: 'Implicit in Brooks' Law is that it applies only to the final phases of a project. The question is, how do you know whether you're in a project's final phases? Because of widespread poor tracking, people think they're at risk from Brooks' Law for significantly longer periods than they really are ' Pointing out that, for Brooks' Law to be true, the amount of effort lost to training must exceed the amount of added work contributed by new staff, McConnell considers Brooks' example of training as absurdly conservative: ' How could the first three people possibly have done enough work in just two months to require a whole month of training for new staff members? The time that existing staff

spends with new staff certainly does take productive time away from the project, and time spent training can be frustrating to existing staff members who are already feeling pressured to complete their own work. But the loss is nothing like the ratio needed for Brooks' Law to be true'.[9] McConnell concludes: 'Controlled projects are less susceptible to Brooks' Law than chaotic projects. Their better tracking allows them to know when they can safely add staff and when they can't Obviously it's beneficial to add staff until some point in the project's schedule, after which adding staff becomes detrimental Every project eventually reaches a point at which adding staff is counterproductive, but that point occurs later than Brooks' law states and in limited circumstances that are easily identified and avoided'.[9]

Whose rule of thumb, then, is the more realistic, Brooks' or McConnell's? The argument based on the notoriously poor accuracy of project estimation and tracking works equally well both for and against Brooks' Law. How should management identify the cut-off point? How to detect the time point after which adding staff becomes detrimental? Due to process delay inherent in project dynamics, by the time the schedule problem is detected, it is already too late to take action. The training effort depends on multiple project and team-specific factors, so that the manager still has to estimate the anticipated cost of training and potential added work. The advantages associated with greater control are not self-evident. According to Lee et al.,[22] exercising less management control, counter to expectations, may actually shorten project duration. Elsewhere, McConnell praises NASA's Software Engineering Laboratory (SEL) as one of the most successful software organizations in the world.[23] We may safely suppose that the SEL management likely uses the best state-of-the-art COTS (commercial off-the-shelf) estimating and tracking tools available. Nevertheless, since 1990, GAO (US General Accounting Office) has identified NASA's contract management as a high-risk area: 'Considerable change in NASA's program cost estimates – both increases and decreases – indicates that NASA lacks a clear understanding of how much its programs will cost and how long they will take to achieve their objectives'.[3]

Another ongoing debate concerns the relevancy of Brooks' Law to free and open-source software (FOSS) projects. Eric Raymond[10] explains that ' Brooks' Law is founded on experience that software bugs tend strongly to cluster at the interfaces between code written by different people, and that communications /coordination overhead on a project tends to rise with the number of interfaces between human beings. Thus, problems scale with the number of communications paths between developers, which scales as the square of the number of developers (N*(N - 1)/2, where N is the number of developers)'. According to Raymond the process of open-source development falsifies the assumptions behind Brooks' Law: ' empirically, if Brooks's Law were the whole picture Linux would be impossible'. Raymond demonstrates that the *bazaar* method, by harnessing the full power of the 'egoless programming' effect, strongly mitigates the effect of Brooks' Law: ' The principle behind Brooks's Law is not repealed, but given a large developer population and cheap communications its effects can be swamped by competing non-linearities that are not otherwise visible'. Raymond introduces 'Linus' Law', which predicts just the opposite relationships between the number of developers and project success: 'Given enough eyeballs, all bugs are shallow'.[10] With a large development and testing community, a problem in software code can be identified more quickly, and a solution produced sooner.

Pointing out that in most FOSS projects, only a few dedicated programmers do most of the code writing, Schweik et al.[24] ask: are FOSS projects more likely to be successful if additional programming support is added to the project? If small core teams do most of the work, then projects with larger teams should not be more successful than projects with small teams. The statistical analysis of the dataset of 30,592 abandoned and 15,782 successful projects found that for each developer added to a project, the odds in favor of the project's success in the growth stage increase by a factor of 1.24.[24] While this finding is consistent with Linus' Law, the authors stipulate that the simple regression model does not provide sufficient evidence for choosing one 'competing theory' over another, since a statistical correlation does not explain the factors that lead FOSS projects to success or abandonment: ' Do more developers lead to project success? Or is it that successful projects attract more developers, in part because of the economic motivations that drive some programmers to participate '. Moreover, ' the relatively flat, modular system of coordination in FOSS projects allows the addition of programmers without too many coordination costs. The real concern in FOSS appears to be not slowing projects down when adding more programmers, but rather, given that most projects have small developer teams, how to get 'more eyeballs' contributing to these projects. Authors conclude that a statistical correlation not necessarily demonstrates a causal relationship and a multivariate model that includes other theoretically-driven covariates is needed'.[24]

Brad Cox's[25] disagreement with Brooks' Law is based on his vision for user-driven development: 'Object-oriented is turning up all over the programmer shortage can be solved as the telephone operator

shortage was solved, by making every computer user a programmer The silver bullet is a cultural change rather than a technological change. It is a paradigm shift; a software industrial revolution that will change the software universe as the industrial revolution changed manufacturing'.

Are Brooks' Law and Linus' Law indeed competing theories of software development? Does Brooks' Law apply only to traditional 'cathedral' type development organizations? Is object-oriented programming indeed led to by the 'software industrial revolution' as envisioned by Brad Cox, solving the programmer shortage 'by making every computer user a programmer'?[25] How relevant is Brooks' Law for agile development? This list of questions can go on.

1.2. Brooks' Law models

Another management gospel attributed to Peter Drucker, states: 'You can only manage what you can measure'.[26] Numerous attempts have been made to quantify Brooks' Law limits through modeling.[13,14,27–31] The first system dynamics model by Abdel-Hamid and Madnik did not fully support Brooks' Law.[27] The authors point out that Brooks' Law has often been endorsed indiscriminately, not only for large-scale, but also for small-scale projects, not only for systems programming projects, but also for applications software. Abdel-Hamid and Madnik[27] clarify Brooks' Law in the following: ' adding more people to a late project always causes it to become more costly but does not always cause it to be completed later unless hiring continues to the end of the project's testing phase'. The authors stipulate that this does not necessarily invalidate Brooks' Law, but rather disqualifies the notion (not implied by Brooks, but hinted at in the writings of others in the literature) that Brooks' Law'' is a universal law of software development.[27]

Does this mean that Brooks' Law is valid only for large, systems programming projects? Hsia et al.[13] criticize the assumptions in Abdel-Hamid and Madnick[27] that: (a) production rate depends solely on the manpower available, so that managers can continuously add new workers as long as they sense a shortage in manpower; and (b) development tasks can be partitioned and there is no sequential constraint among them. Hsia et al.[13] argue that if development tasks have to be performed sequentially, then adding to the workforce may not speed up production, since there may be not enough tasks available for all the workers to work on. They also point out that, since it is not always easy to obtain approval for additional manpower, in reality the workforce is usually added to only once during a project's lifetime. In their opinion, an optimal time for adding workers without delaying a project ranges from one-third to halfway through the project's lifetime, so that if managers cannot make a timely decision on project re-staffing prior to halfway into the project, the project has a high probability of being delayed.

The mathematical model by Richard Stutzke[30] includes: f – fractional increase in staff, r – time (workdays) remaining for project completion, a – assimilation time (workdays), m – mentoring cost, the fraction of a staff member's time spent mentoring one new hire, E_a – assimilation effort, E_u – useful effort delivered to project, and E_e – expended effort. According to calculations based on this model, staff can be added to a project to stay on schedule, subject to two constraints: if (a) $r > a^1$ (effective assimilation time); and (b) $f <= 1/\mathrm{m}$.[30] As possible refinements to the model Stutzke[30] considers including increased mentoring effort with mentoring workload, decay in the learning curve, and decrease in productivity as staff size increases (communication overhead), but thinks that these effects are not significant.

Williams et al.[31] used Stutzke's model to see whether Brooks' Law holds in pair programming practice. Their results indicate that since effective assimilation time and mentoring time are reduced due to pairing, adding manpower to a late project yields productivity gains earlier. Due to quicker assimilation, more work is achieved in the remaining days: a project manager who would have to add 11 people to complete the project on time, would need to add only nine if the team practices pair programming (saving the wages of two systems analysts for 121 days, with each analyst making an average of \$231/day).[31]

Among the assumptions in Stutzke[30] are: the work remaining is well understood; additional effort needed to complete the project is known; all new workers are competent, and must learn only project-specific information; each new worker is assigned to one experienced person; mentoring is uniformly distributed to a subset of the experienced staff; new workers learn at a uniform rate; original and augmented teams have the same productivity; and variations in productivity due to staff size, e.g., communication overhead or overtime worked by both experienced and new employees, are ignored.

What if even one of the above assumptions does not apply to the actual situation? What if the work remaining is not well understood and additional effort required for completion is not known? What if not all the new workers are competent or learn at different rates? What if the manager cannot provide each new worker with a personal mentor and there are simply not enough experienced people to distribute mentoring duty uniformly among them?

What if the original and augmented teams would have different productivity? Stutzke[30] views communication overhead as a second-order effect that cannot be directly quantified: is it sufficient justification to ignore productivity lost due to increased staff? Although Stutzke[30] provides the logical and realistic equations for calculating training effort (endorsed by Brooks[5]), they are not sufficient for concluding whether Brooks' Law applies.

Raymond Madachy[29] provided the only model capturing core causal relationships between productivity, training, and communication overheads for years used to demonstrate the pitfalls of project dynamics. While testing this model, we found its response to stepwise changes of input is not consistent. A model may be considered robust if it continues to operate despite abnormalities in input, at least within the reasonable scope of input assumptions. Our efforts to formulate a robust expression of Brooks' Law resulted in a ProjScout™ model. In present article the results of testing both models and more advanced experiments, performed to evaluate the conceptual and operational validity of the ProjScout™ model, are presented. Section 1.3 describes the model by Madachy[29] (thereafter referred to as MBLM). Section 2.1 provides the results of MBLM testing. Section 2.2 describes the basic ProjScout™ model. In Section 2.3, the results of ProjScout™ tests using the same scenario and assumptions as in MBLM tests are provided. Section 2.4 describes advanced simulation experiments with the basic ProjScout™ model: the effects of team size and experience (Section 2.4.1), the effect of team investment in training (Section 2.4.2), and the effect of variations in nominal productivity, difference in productivity between experienced and new workers and the effect of communication entropy (Section 2.4.3). Section 2.5 describes experiments with various staffing scenarios for a sample team. Section 3 discusses boundaries, limitations, added value, and possible applications of ProjScout™.

1.3. Description of MBLM

The following description of MBLM is from the latest published version.[29] The model is based on the following assumptions: new personnel require training by experienced personnel to come up to speed; having more people on a project entails greater communication overhead; and experienced personnel are, on average, more productive than new personnel. MBLM is built in two connected flow chains representing software development and personnel (Figure 1).

The software development chain assumes a level of *requirements* transformed into *developed software* at the *software development rate*. The level of *developed software* represents progress made on implementing the requirements. Project completion is when *developed software* equals the initial *requirements*. Software size is measured in function points, and the development rate is in function points/day. The *software development rate* is determined by the levels of personnel in the system: *new project personnel* who come onto the project at the *personnel allocation rate*, and *experienced personnel* who have been assimilated (trained) into the project at the *assimilation rate*.

Software development is constrained by several factors: the *nominal productivity* of a person, the *communication overhead %*, and the effective number of personnel. The effective number of personnel equals the *new project personnel* plus the *experienced personnel* minus the amount of *experienced personnel needed for training* the new people. The *communication overhead %* is expressed as a non-linear function of the total number of personnel that need to communicate ($0.06 \times n^2$). The *experienced personnel needed for training* is the training overhead percentage as a fraction of full-time equivalent experienced personnel. The default of 0.25 indicates one quarter of an experienced person's time is needed to train a new person until he/she is fully assimilated.

The bottom structure for the personnel chain models the assimilation of *new project personnel* at an average rate of 20 days. In essence, a new person is trained by one fourth of an experienced person for an average of 20 days until they become experienced in the project. The *nominal productivity* is set to 0.1 function points/person-day, with the productivities of new and experienced personnel set to $0.8 \times$ *nominal productivity* and $1.2 \times$ *nominal productivity*, respectively, as a first-order approximation.[29]

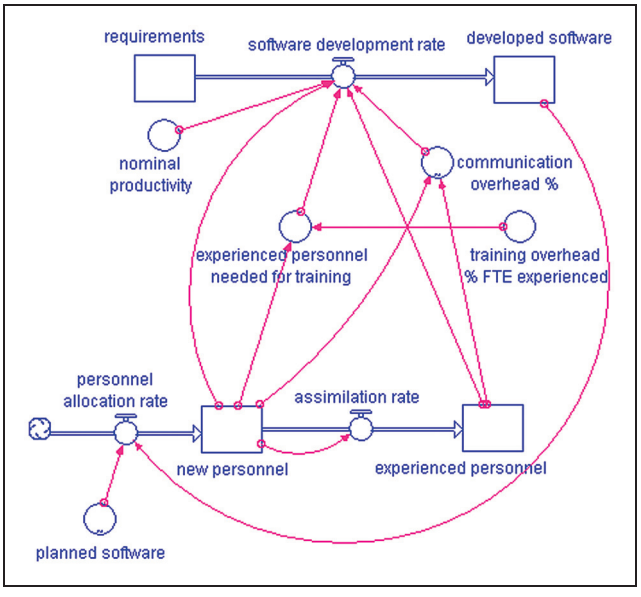

Figure 1. The MBLM stock-flow diagram.[29]

The following description of MBLM simulation output (Figure 2) is from Madachy[29]. Steady-state conditions are obtained in the model when no new people are added (curve #1). The default behavior of the model shows a final completion time of 274 days to develop 500 function points with a constant staff of 20 experienced personnel. The model is activated when more people are added at the personnel allocation rate. The rule for staffing perturbation implemented through an 'IF, THEN' statement, is: add more people if the gap between planned and completed product is greater than the scheduled threshold of 15%, after 30% of the project time has elapsed. With an extra staff of five people (curve #2), the development rate nosedives, and then recovers after a few weeks slightly to overtake the default rate, and actually finishes sooner at 271 days. However, when 10 people are added, the project as a whole suffers (curve #3). The initial productivity loss takes longer to stabilize, the final productivity is lower with the larger staff, and the schedule time period extends to 296 days. The plunge and smooth recovery seen on the two curves are the training effect. The extra staff gains in the first case are greater than the communication losses, but going from 20 to 30 people in the second case entails a larger communication overhead, as compared with the potential productivity gain of having more people.[29] Madachy concludes:

> This model of Brooks' Law demonstrates that the law holds only under certain conditions (Brooks did not mean to imply that the law held in all situations). There is a threshold of new people that can be added until the schedule suffers, showing the tradeoff between adding staff and the time in the lifecycle The increased productivity must persist long enough to outweigh the initial productivity drains The model can be used to experiment with different scenarios to quantify the operating region envelope of Brooks' Law. A specific addition of people may be tolerable if injected early enough, but not later. The project time determines how many can be effectively added.
>
> The model uses simplified assumptions and boundaries, and can be refined in several ways. The parameters for communication overhead, training overhead, assimilation rate and other formulations of personnel allocation are also important for a thorough sensitivity analysis. A myriad of different combinations can and should still be tested Based on the insight provided, we may now clarify Brooks's Law. *Adding manpower to a late software project makes it later if too much is added too late* This model is a microcosm of the system dynamics experience. Simplifying assumptions are made that coincide with the simple purpose of this model. The reader may need some faith at first since rationale for the full model formulation was not yet given.[29]

While MBLM is indeed a microcosm of the system dynamics experience, the question is, how fairly does it represent a microcosm of Brooks' Law? Indeed, we 'need determining the adequacy of the model before experimenting with it', that is, before engaging in the exploration of 'a myriad of different combinations'.

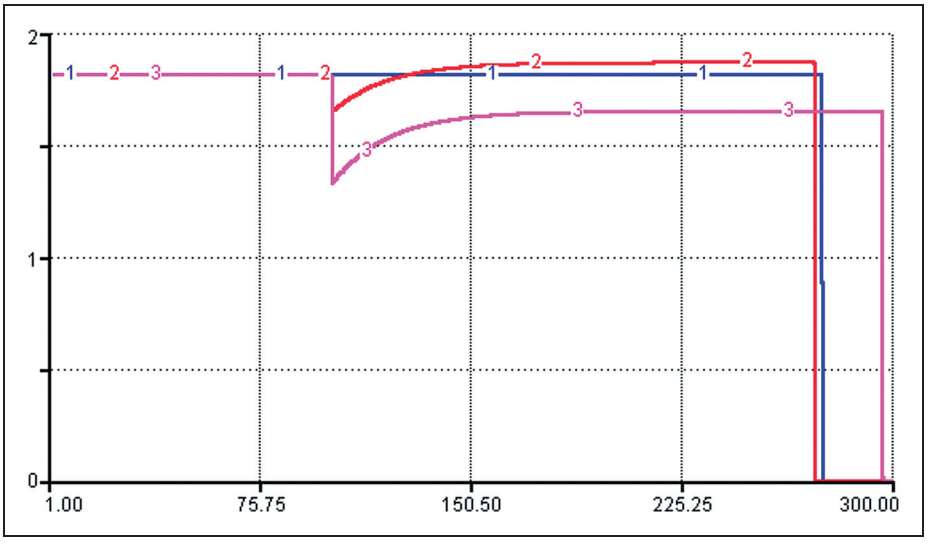

Figure 2. MBLM. Sensitivity of *software development rate* (function points/person-day) to varying personnel allocation pulses according to Madachy.[30a] 1: no extra hiring; 2: add five *new personnel*; 3: add 10 *new personnel* to 20 *experienced personnel* on the 100th day.

The validation process includes verification of the model's boundaries and structure, parametric and dimensional consistency, and behavioral tolerance of extreme conditions.[1,29,32,33] Sensitivity analysis is used to discover how sensitive the models are to changes in the values of the input variables. Is the model robust in the face of extreme variations in input assumptions, and does it exhibit logical behavior when selected parameters are assigned extreme values?[34,35] If the answers to these questions are negative, any painstaking validation of the model boundaries, structure, parameters, and dimensional consistency is superfluous. In the end, we want to be sure we navigate a microcosm of Brooks' Law rather than a microcosm of a model.

2. Results

2.1. Testing MBLM

This section provides results of MBLM testing. Our test of MBLM according to Madachy[29] showed that with an extra staff of five people, the development rate nosedives sharply and then increases to 2.13 function points/day, enabling project completion 19 days sooner. When 10 people are added, the development rate after a smaller plunge quickly increases to 3.14 function points/day (72.5% increase comparing to default rate), enabling project completion 63 days earlier. Thus instead of expected Brooksian response, the MBLM demonstrates a plain 'dose-effect' schedule acceleration. The communication overhead was unresponsive to the number of added people, rising in both variants from 24 to 54%. Checking dynamics of the personnel chain found that with five people added to the initial 20, the experienced personnel stock increased to 39 people and 10 people resulted in the personnel stock of 58 people! Since such a 'population explosion' indicates an apparent calculation error, for further testing we used an earlier MBLM version according to Madachy and Boehm[30a], Madachy and Tarbet[30b] and recently Wu and Yan[30c], which produced behavior shown in Figure 2 and where staffing perturbation was induced using the PULSE function without any schedule-produced feedback. Only varying numbers of initial personnel and staffing pulses are used as input parameters. The results of the MBLM's extreme-value test are shown in Figure 3(a). As described in Madachy,[29,30a] after a pulse of five people (curve #2), the *software development rate* nosedives and then recovers after a few weeks to slightly overtake the default rate. When 10 people are added, the overall project suffers (curve #3).

Logically, the further increase in staffing pulse should increase the distraction of the experienced personnel from the production and result in a deeper plunge and longer recovery of development rate. However, a stepwise increase in staffing pulse does not produce this expected trend. Adding 15 rookies (curve #4) brings about a smaller plunge and a quicker recovery of the *software development rate*, stabilizing at a higher level than before the addition of workforce. Adding 20 rookies (curve #5), or more (not shown), causes a smaller plunge and a quicker recovery of the production rate, stabilizing at a much higher level. Instead of the paradoxical effect of Brooks' Law, the observed schedule compression reflects a rather plain 'dose-effect' production gain in response to increased manpower.

While assigning extreme value to input is the essence of this test, one can argue that such levels of staffing intervention – up to 100% of the original personnel – is not typical. Even if MBLM is not robust it could nevertheless be sufficiently representative, demonstrating logical response when inputs are within reasonable limits. To test MBLM sensitivity to a staffing pulse of 25% of the initial workforce – the level that causes production gain, 4–8 rookies were added to teams of 16, 20, 24, 28, and 32 veterans (Figure 3(b)). Surprisingly, the model responded as described in Madachy[29] only in one case – when five rookies were added (curves #2). Other proportional combinations of initial workforce and staffing pulses produced various, rather puzzling responses. Adding four rookies to 16 veterans (curve #1) was most counterintuitive if we compare it to the steady-state condition: after assimilating four rookies, the resulting 20-person team can complete the project 13 days ahead of schedule, compared to a team of 20 veterans, working from the start without distraction. Six rookies added to 24 veterans cause a deeper plunge (compare curves #2 and #3) and indeed makes project 16 days later. However seven rookies added to an 28 veterans (curve #4),) cause a smaller plunge and make project 5 days earlier. And at last, adding eight rookies to 32 veterans (curve #5), causes no production drop. Contrary to expected, added workforce simply boosts the production. Then the production rate continues to increase reflecting rookies' assimilation and enabling to complete project much earlier.

A high degree of sensitivity is a warning to interpret the results of the model with circumspection, particularly because many of the input variables themselves will have been estimated and therefore are subject to error.[2] Barlas[35] demonstrated that a behavioral sensitivity test, originally suggested by Forrester and Senge[32] as a validity test, could detect major structural flaws in a model despite the fact that the model generates highly accurate behavior patterns. In both above tests the MBLM produces neither a plain 'dose-effect', nor characteristic Brooksian response. Only the staffing pulses used in Madachy[29,30a] have produced a

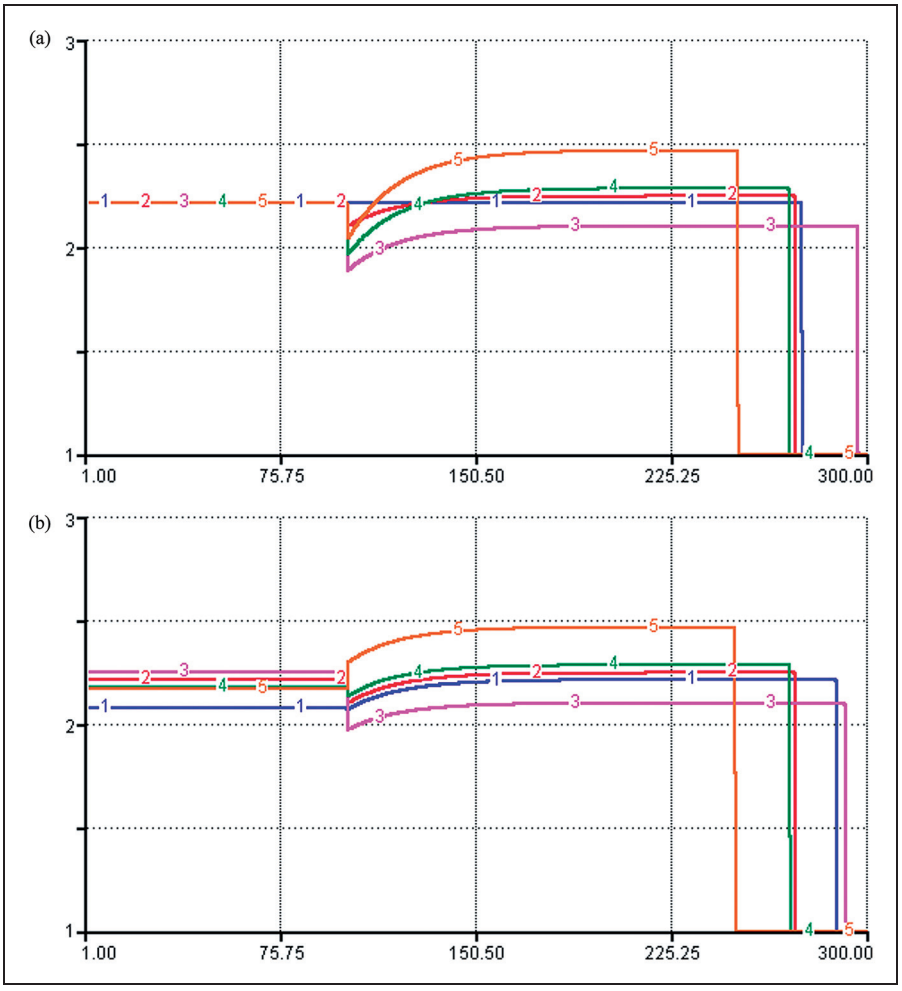

Figure 3. MBLM. *Software development rate* (function points/person-day). (a) Extreme value test. 1: no extra hiring; 2: add five; 3: add 10; 4: add 15; 5: add 20 *new personnel* to 20 *experienced personnel* on the 100th day. (b) Proportional value test. 1: add four to 16; 2: add five to 20; 3: add six to 24; 4: add seven to 28; 5: add eight *new personnel* to 32 *experienced personnel* on the 100th day. All parameters are as in Figure 2.

response that can be interpreted as an effect of Brooks' Law. It means that either Brooks' Law works only within an extremely narrow window of inputs or, more plausibly, that MBLM does not sufficiently represent it.

To figure out the possible source of error we looked at other outputs of the model. According to Brooks[11]: 'The added burden of communication is made up of two parts, training and intercommunication... training cannot be partitioned, so this part of the added effort varies linearly with the number of the workers Intercommunication is worse. If each part of the task must be separately coordinated with each other part, the effort increases as n(n-1)/2. Three workers require three times as much pair wise intercommunication as two; four require six times as much as two'. Correspondingly, the production rate plunge and recovery should reflect both an effect of training and a higher communication overhead. While the MBLM[30a] successfully produced dynamics of the personnel stock, *assimilation rate and experienced personnel needed for training* in both tests, the way of introducing the communication overhead made it non-responsive to the team size. In the extreme-value test (Figure 4(a)), the communication overhead increased when 10 people were added as compared to five (curves #1, 2, and 3), but did not reflect the further step-wise increase of team size (curves #3, 4, and 5). The steady-state condition runs in the proportional-value test (Figure 4(b)) show that the increase of *communication overhead* does not scale proportionally with the number of communications paths between developers. While adding four to 16 people increases the communication overhead (curve #1) to 24% and adding five to 20 people increases it to 37.5% (curve #2), the 25-percent staffing pulse of original 24, 28 people and 32 people (curves #3, #4 and #5) results in the same 54% communication overhead. It means that 30-people team has the same communication overhead

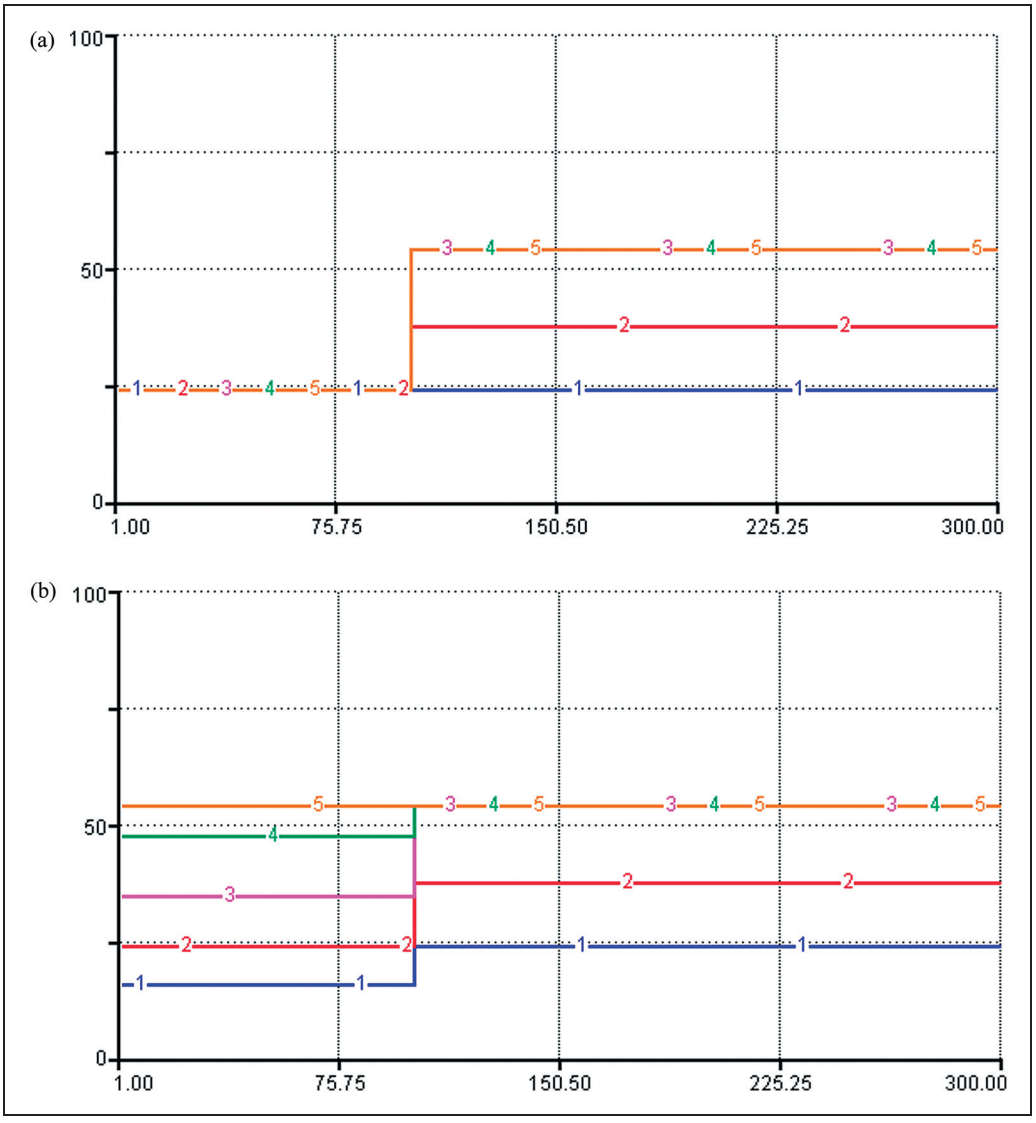

Figure 4. MBLM. Sensitivity of communication overhead to varying personnel allocation pulses. (a) Extreme value test. (b) Proportional value test. All variants are the same as in Figure 3.

as 35 or even 40 people (curve #5) where communication overhead stayed flat before and after the adding people. These results suggests that one MBLM's flaw is related to the method of introducing the communication overhead.

According to Raymond[10]: ' Brooks' Law (and the resulting fear of large numbers in development groups) rests on a hidden assumption: that the communications structure of the project is necessarily a complete graph, that everybody talks to everybody else. But on open-source projects, the halo developers work on what are in effect separable parallel subtasks and interact with each other very little; code changes and bug reports stream through the core group, and only within that small core group do we pay the full Brooksian overhead'. In a conventional project, reflected in Stutzke's model,[30] even a 20-person team is usually broken down into four or five groups. Project veterans train the newcomers and training-related communication within such groups is reasonably limited. A more realistic assumption would be to account for the organization communication scale or entropy *B*-factor separately, without association with training. This and other considerations have been reflected in the ProjScout™ structure.

2.2. Description of basic ProjScout™

Basic ProjScout™ is comprised of two stock flow chains: Production and Personnel (Figure 5).

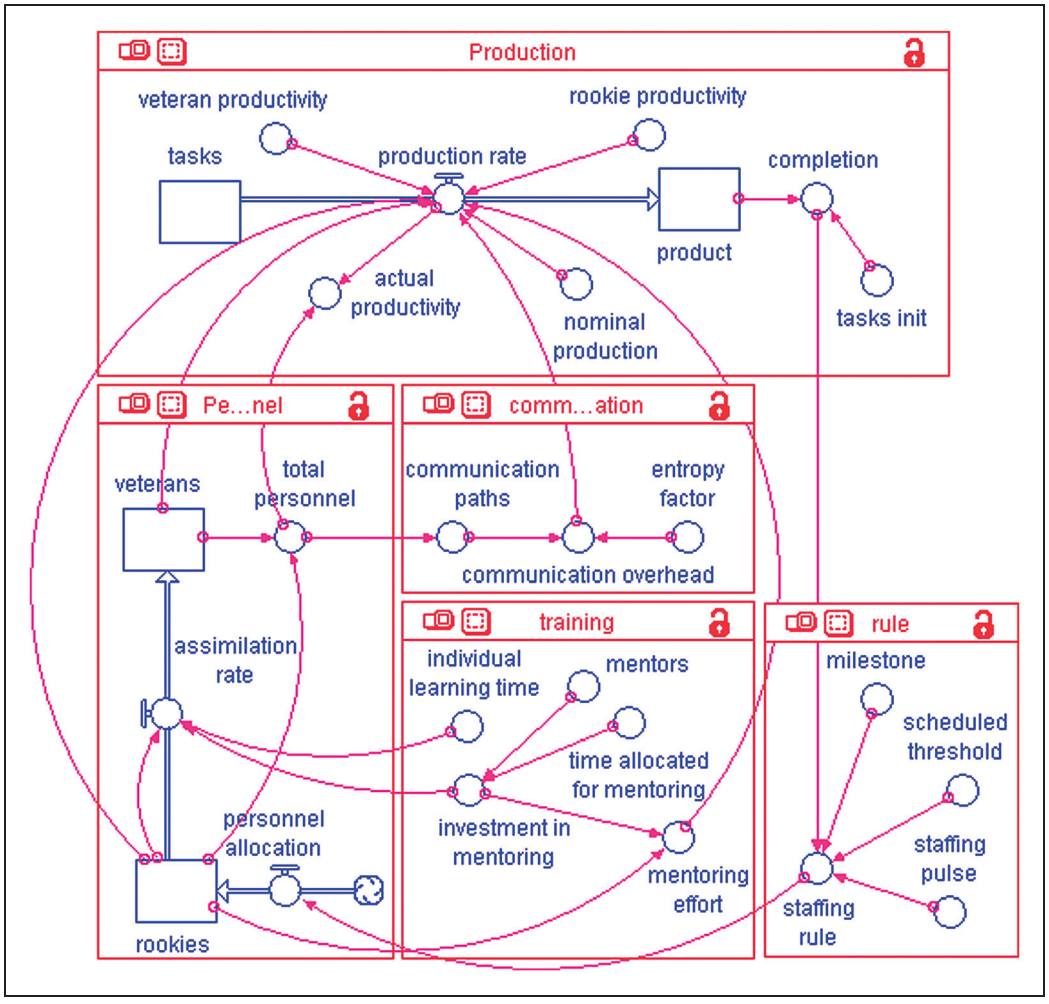

Figure 5. The stock-flow diagram of the ProjScout™ model (Stella simulation software from ISEE Systems, Inc.).

The Production chain assumes a stock of *tasks* to be transformed into a stock of *product* (based on Madachy[29,30a] and Stutzke[30] with changes summarized in Table 1):

$$tasks(t) = tasks(t - dt) + (-production\ rate) \times dt \quad (1)$$

$$product(t) = product(t - dt) + (production\ rate) \times dt \quad (2)$$

at the

$$production\ rate = nominal\ production \times (1 - communication\ overhead) \times ((rookie\ productivity \times rookies) + (veteran\ productivity \times (veterans - mentoring\ effort))) \quad (3)$$

The *dt* was set to 0.25. The *production rate* is a group daily effort measured in tasks (i.e. function points) per team per day. Dimensionless *veteran productivity* and *rookie productivity* are modifiers of *nominal production* modeled as the daily effort of one person (task/person-day). The *communication overhead* is modeled as a percentage of a product of an arbitrary value of the communication *entropy factor* and total number of *communications paths* in the team (In basic model the sub-team communication effort and the overall organization entropy are accounted for together):

$$communication\ overhead = entropy\ factor \times communication\ paths 100 \quad (4)$$

where

$$communication\ paths = total\ personnel \times (total\ personnel - 1) 2 \quad (5)$$

The training-related effort is accounted for as *mentoring effort* (person-day):

$$mentoring\ effort = investment\ in\ mentoring \times rookies \quad (6)$$

where

$$investment\ in\ mentoring = mentors \times allocated\ for\ mentoring \quad (7)$$

Table 1. The summary of basic ProjScout™ compared to Madachy[29,30a] and Stutzke[30]

Variables and assumptions	Reference models	ProjScout™
Total workforce	The effective number of personnel is *new personnel* plus the *experienced personnel* minus the *amount of experienced personnel needed for training* the new personnel [29,30a].	*rookies* plus *veterans*.
Initial workforce	*experienced personnel*[29,30a].	*rookies* plus *veterans*.
Learning time	The *individual learning time*[30].	The *individual learning* time is a number of workdays required for rookies' assimilation into the project without team investment in training.
Allocated for mentoring	The *mentoring cost*—a fraction of staff member's time, spent mentoring one new person[30].	A fraction of veterans' time reallocated away from production to mentor *rookies*.
Training overhead	A fraction of an equivalent full-time *experienced personnel*. A new person is trained by 1/4 of an experienced person for an average of 20 days[29,30a].	The *mentoring effort* (person-day) is a product of number of *rookies* and *team investment in mentoring*, a product of the number of *mentors* and *allocated for mentoring*.
Communication overhead	A custom graph input[29,30a].	A percent of a product of *communications paths* (*total personnel* × (*total personnel* -1)/2) and assumed value of *entropy factor* of total *communications paths*.
Planned work completion	A custom graph input[29,30a].	*completion* = PCT(*product* / *tasks init*)
Staffing perturbation	IF((*developed software* − *planned software*) <−75 and TIME <112) THEN *staffing pulse* ELSE 0[29] or PULSE (*staffing pulse*, 100, 999)[30a]	IF(*completion* < *scheduled threshold* AND TIME = *milestone*) THEN PULSE (*staffing pulse, milestone*, 0) ELSE 0

where *mentors* is the number of veterans charged with mentoring rookies, limited by availability, organization policy, worker's physical proximity, and other factors; and *allocated for mentoring* is a fraction of veterans' time (dimensionless) allowed for mentoring away from production, which may vary from 0.125 (1 hr a day) to 1.0 (full time).

The *Personnel* chain assumes a stock of *rookies* to be transformed into a stock of *veterans* at the *assimilation rate* (person/day):

$$\begin{aligned} &assimilation\,rate = rookies \\ &\quad \times investment\,in\,mentoring individual\,learning\,time \end{aligned} \tag{8}$$

where *individual learning time* is the average number of workdays required for rookies' assimilation into the project.

While there are various staffing schedules and additional in- and outflows can be applied, for the sake of simplicity the flat staffing schedule was applied in basic ProjScout™, assuming all personnel are available from the start of the project. Therefore,

$$personnel\,allocation = staffing\,rule \tag{9}$$

where

$$\begin{aligned} staffing\,rule = &\text{IF}\,(completion < scheduled\,threshold \\ &\text{AND TIME} = milestone)\,\text{THEN PULSE} \\ &(staffing\,pulse\,milestone\,0)\,\text{ELSE 0} \end{aligned} \tag{10}$$

and staffing pulse is a number of added *rookies, milestone* (day) and *scheduled threshold* (%) are variables.

The *nominal production* (task/person-day), dimensionless *veteran productivity*, *rookie productivity*, entropy factor, *allocated for mentoring* (workday/day), individual learning time (days), and *mentors* (persons) are assumed project-scale variables. Two outputs are introduced to provide feedbacks:

$$completion = \text{PCT}\,(product\;\; tasks\;init) \tag{12}$$

showing overall work progress and providing input to the *staffing rule*, and

$$actual\,productivity = production\,rate total\,personnel \tag{13}$$

where

$$total\,personnel = rookies + veterans \tag{14}$$

providing input for calculating *communication overhead*.

2.3. Testing ProjScout™

The ProjScout™ model has been calibrated within following range of input: 0.05–0.2 task/day *nominal production*, 1.0–5.0 *veteran productivity*, 0.2–1.0 *rookie productivity*, 5–60 days of *individual learning time*, 0.125

(1 hr a day) to 1.0 (full-time mentoring) of time *allocated for mentoring*, *mentors* from one to all available veterans and 0.01–014 communication *entropy factor*. For the sake of comparison, the ProjScout™ was tested using the same scenario and assumptions as MBLM: 1.2 *veteran productivity*, 0.8 *rookie productivity*, 0.06 communication *entropy factor*, and 300 workdays of the estimated schedule. The *nominal production* was set to 0.1 task/person-day. To justify the staffing intervention the default project size (*tasks init)* was set to 1,000 tasks and *scheduled threshold* to 22%. Those numbers assume that 27–28 *veterans* can provide the *production rate* of 3.3 task/team-day required to meet the schedule.

The steady-state run showed that, as expected, 20 *veterans* could not meet the schedule (Figure 6(a), curve #1), therefore justifying the addition of manpower. According to the staffing intervention rule, when an arbitrary schedule delay is detected at the 100th day of the project, *rookies* are added to the original team. In the extreme-value test (Figure 6(a)), 5–20 *rookies* are added to the initial 20 *veterans*. In all cases, adding people causes a proportional drop in *production rate* followed by gradual recovery, stabilizing at various but always higher levels than before staffing correction (Figure 6(a) curves #2–5). The highest relative production gain (15.5%) from 2.13 to 2.46 task/day is achieved with five added *rookies* (curve #2). Adding 10 or 15 *rookies* (curves #3 and 4) brings about a relatively smaller production increase, whereas the addition of 20 *rookies* (curve #5) significantly slows *production rate* down.

The characteristic Brooksian response of the team *production rate* to stepwise increase of personnel stock is summarized in Figure 6(b). Whereas the team *production rate* non-linearly increases with 5–10 *rookies* added and drops only with a further increase in staffing pulse (curve #1), the *actual productivity* always suffers, declining linearly with the total team size (curve #2). Increase of team size above 50% may be counter-productive, considering the cost of the achieved production gain. Increase of personnel above 75% slows production rate down. No team under the model's assumptions can meet the schedule, completing only 64, 69, 71, 71, and 67% of work before the deadline correspondingly. The team of 20 veterans without extra help will need an additional 170 days, with five added rookies –128 days, with 10 rookies –109 days, and with 15 rookies – 108 days to complete the project. Adding 20 rookies delays completion on 20 days more, so 25- and 40-person teams would complete the project on the same, 428th day.

The increasing of the assumed *veteran productivity* from 1.2 to 2.0 and decreasing *rookie productivity* from 0.8 to 0.2 enables all resulting teams to meet schedule (Figure 6 (c)). The highest absolute production gain is achieved with 10 added rookies (curve #3). Teams of 25- and 35-people compete project almost synchronously and later (curves # 2 and # 4) comparing to 30-people team; and 40-people team (curve #5) completes project almost at the same time as 20 people (curve #1).

Testing ProjScout™ sensitivity (Figure 7) shows that in the steady state the *production rate* increases non-linearly with increase of initial personnel. A staffing pulse of 25% of initial workforce amplifies this trend: whereas adding 4–7 *rookies* to 16–28 *veterans* improves *production rate* diminishingly, adding eight *rookies* to 32 *veterans* is counter-productive (Figure 7(a), curve #5). A 40-worker team is less productive, than a 35- or 30-worker team (Figure 7(a), curves #3 and 4). The maximum production gain is achieved with 10–15 *rookies* added. The actual effect of added workforce is determined by specific values of productivity and production overheads. Since the *average productivity* always drops with increase in *total personnel* (Figure 7(b), curves #3 and 4), the cost of production gain has to be taken into account. Under the model's assumptions and considering the cost of workforce, adding five people to an initial team of 20 experienced workers seems to be the preferable choice. No team can meet the schedule completing only 59, 69, 75, 78, and 76% of work before the deadline, correspondingly: a 20-person team needs an additional 194 days, 25 – 128 days, 35 – 82 days; 30- and 40-person teams need an additional 93 days to complete the project.

With assumed 10-times difference between productivity of veteran and rookie personnel the production gained by adding 25% of original workforce is decreasing with team size (Figure 7 (c)). The team of 20-people where four rookies were added on 100th day cannot meet the schedule comparing to 20-veteran team working from the beginning (compare curves #1 in Figure 6 (c)) and Figure 7 (c)). Adding five to 20 provides highest relative gain in production rate (curve #2 in Figure 7 (c)) and 35-people team provide highest schedule acceleration (curve #4), 40 people complete the project 4 days later (curve #5) comparing to 35 people.

The dynamics of *mentoring effort* (person-day) and *communication overhead* reflect the stepwise increase in both the number of trainees and total personnel. Since the project starts with all personnel being *veterans*, the *mentoring effort* (Figure 8) comes into play only after staffing intervention. The daily *mentoring effort* is between five and 20 person-days, depending on the number of *rookies* added, and gradually decreases, as they get assimilated into the project.

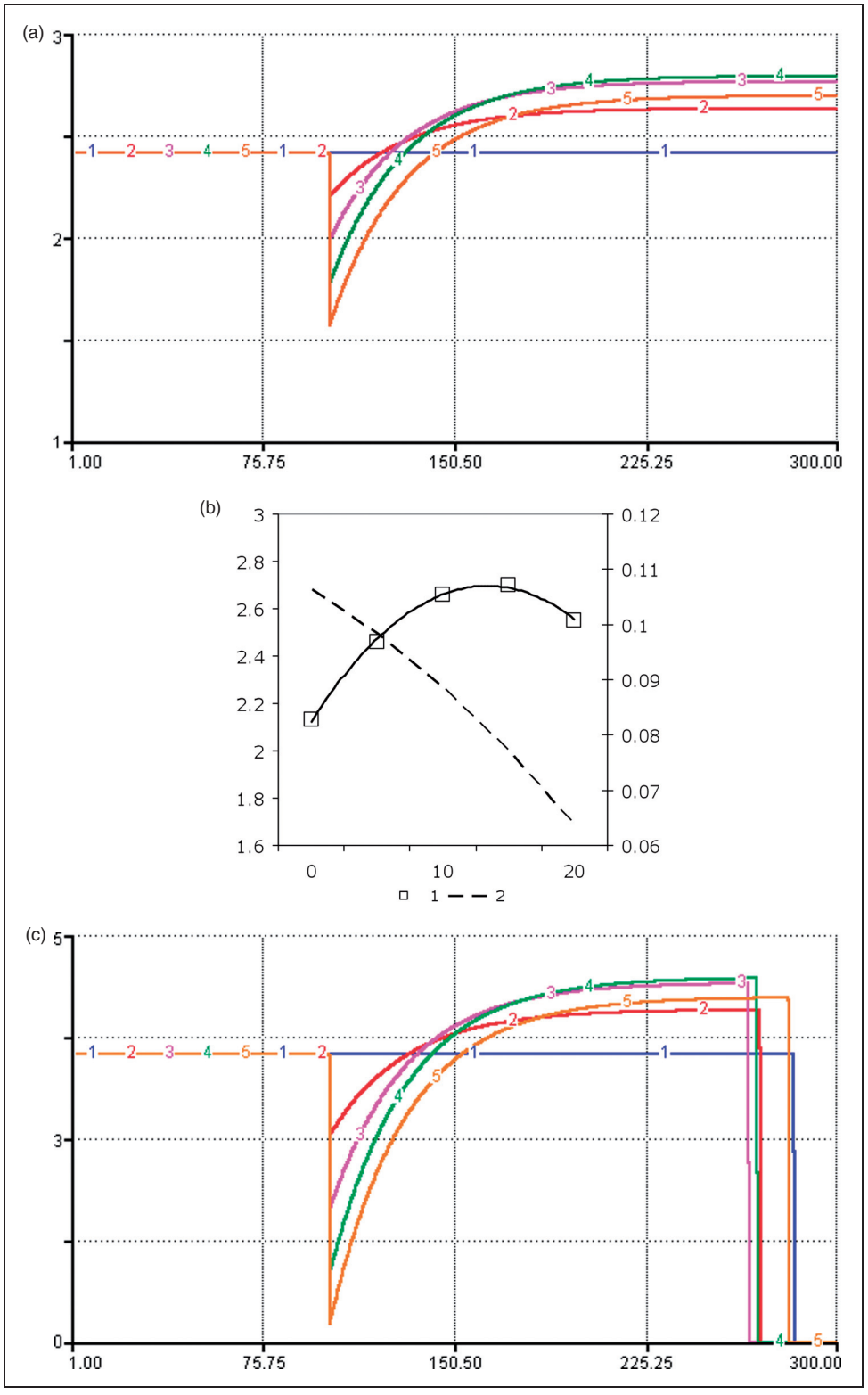

Figure 6. ProjScout™. Extreme value test. Production rate. (a) All variants and parameters are the same as in Figure 3(a), *scheduled threshold* = 22%. (b) Effect of added manpower. *X*-axes – staffing *pulse* (*rookies*); left *Y*-axes – *production rate* (task/team-day), right *Y*-axes – *actual productivity* (task/person-day). (1) *production rate*; (2) *actual productivity*. *entropy factor* – 0.06, *mentors* – 4, *allocated for mentoring* – 0.25 workday/day. (c) *veteran productivity* – 2.0, *rookie productivity* – 0.2, *scheduled threshold* – 36% (to trigger adding workforce in all variants). All other parameters are as in (a).

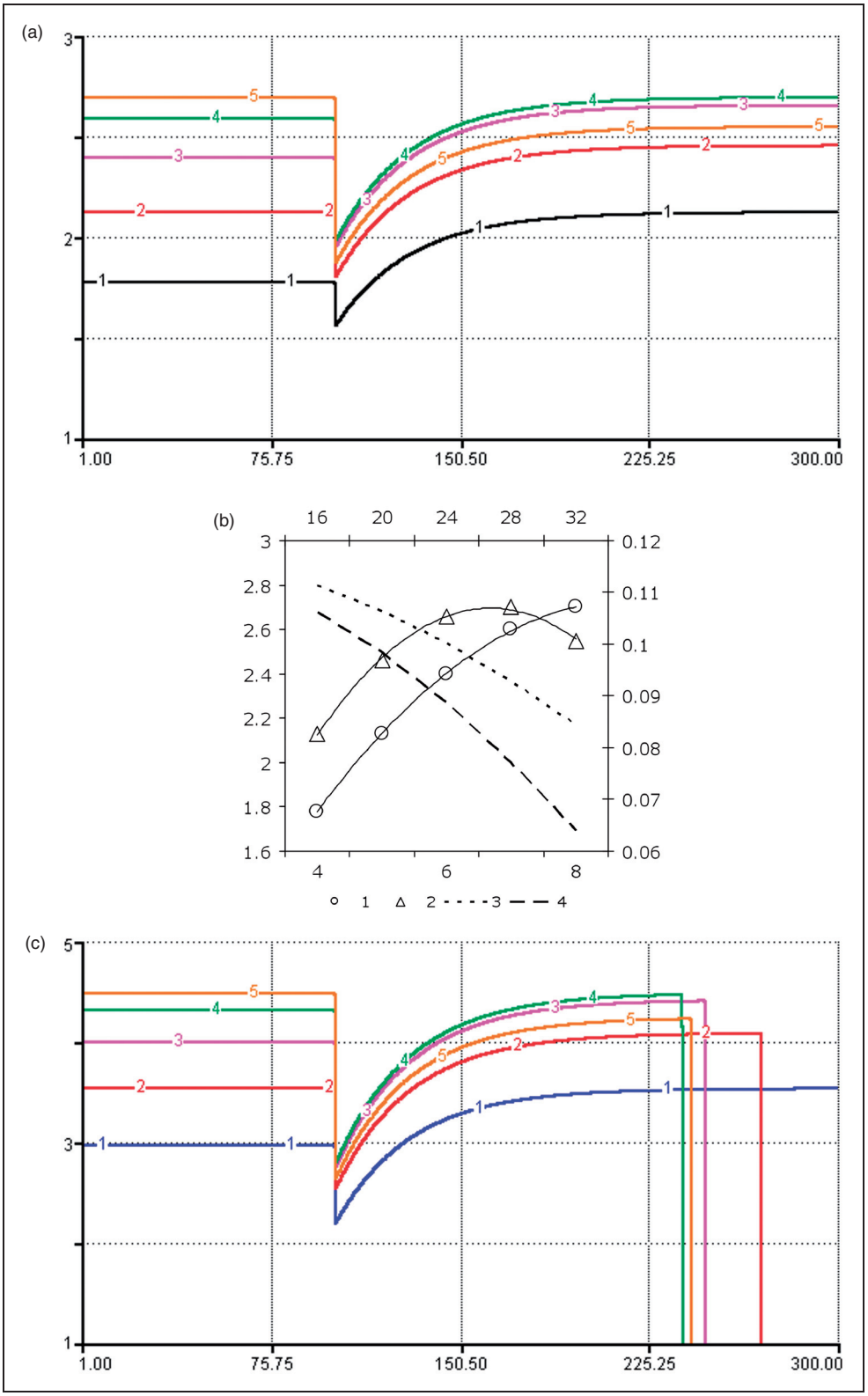

Figure 7. ProjScout™. Proportional value test. Production rate. (a) All variants and parameters are the same as in Figure 3(b), *scheduled threshold* = 27%. (b) Effect of initial team size and staffing pulses. Upper *X*-axes – initial team (*veterans*); lower *X*-axes – staffing pulse (*rookies*); left *Y*-axes – *production rate* (task/team-day), right *Y*-axes – *actual productivity* (task/person-day). (1) *production rate* before and (2) after the 100th day; (3) *actual productivity* before and (4) after the 100th day. (c) *veteran productivity* – 2.0, *rookie productivity* – 0.2, *scheduled threshold* – 45%. All other parameters are as in (a).

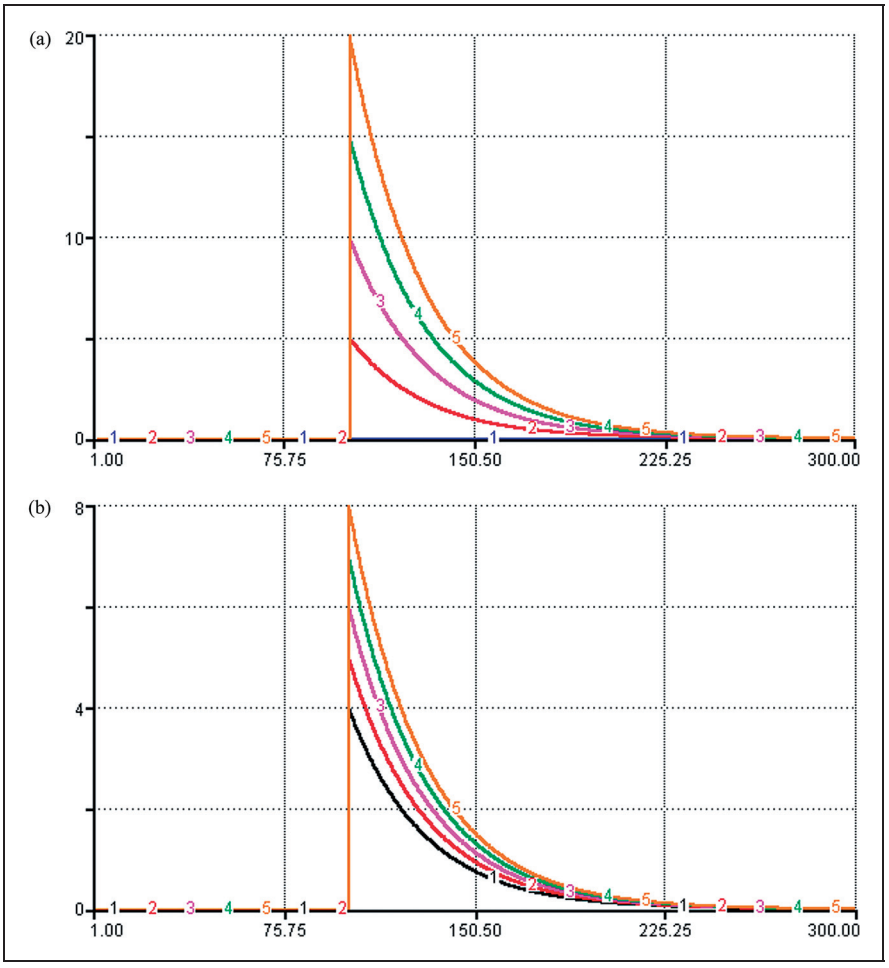

Figure 8. ProjScout™. Sensitivity of mentoring effort to staffing pulses. (a) Extreme value test. (b) Proportional value test. All variants and parameters are as in Figures 6(a) and 7(a).

The dependence of *communication overhead* on team size is the same in both the extreme-value and sensitivity tests (Figure 9). In the extreme-value test, before the 100th day it is 11.4% in all variants (Figure 9(a)), since the starting team is the same. After adding people, the *communication overhead* increases non-linearly, from 18% for a 25-person team to 47% for a 40-person team. The different level of *communication overhead* before the staffing intervention (7, 11, 17, 23, and 30%, correspondingly) in the proportional-value test (Figure 9(b)) reflects the different size of the initial team. After adding people, *communication overhead* increases to 11, 18, 26, 36, and 47%, correspondingly, reflecting increase in the team size.

The above tests show that ProjScout™ logically responds to both initial team size and staffing intervention and remains stable with stepwise changes of input parameters. ProjScout™ successfully emulates the production and personnel dynamics in full compliance with Brooks' Law.

2.4. Exploring ProjScout™

The following experiments were performed for further evaluation of the conceptual and operational validity of the basic ProjScout™ model: testing the dependence of the schedule and production on the number and experience of personnel, the effect of team investment in training, and the role of nominal values of productivity and communication entropy.

2.4.1. Effect of team size and experience.

At steady state, with no communication and training overheads taken into account and no perturbations caused by staffing intervention, the *production rate* linearly increases with team size (Figure 10(a)) for both veteran (curve #1) and rookie teams (curve #2). At *nominal production*, 28 *veterans* (Figure 10(b), curve #1) or 42 *rookies* (curve #2, $[y = 0.16x + 1.44]$) can complete the work on schedule. Correspondingly, the actual resulting *production rate* is determined by the ratio of *rookies* to *veterans* in the team. With a change of the *rookie*/*veteran* ratio

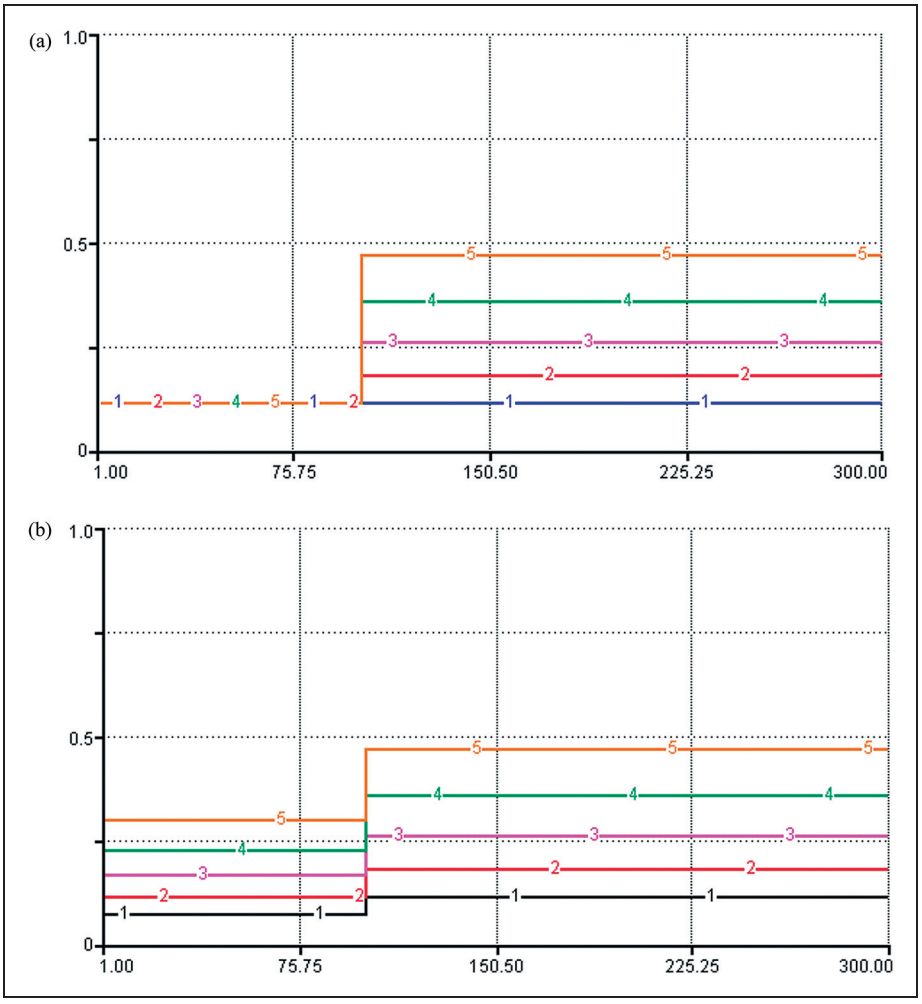

Figure 9. ProjScout™. Sensitivity of communication overhead to staffing pulses (maximum value of 1.0 of Y axis corresponds to 100%). (a) Extreme value test. (b) Proportional value test. All variants and parameters are as in Figures 6(a) and 7(a).

from 33/2 to 25/10 (Table 2), the *production rate* linearly increases from 2.88 to 3.2 and work *completion* from 86.4% to 96%. With the default productivity values, a veteran would be only 40% (1.2) more productive than a rookie (0.8). This difference was used only for the sake of comparability and seems to be strongly underestimated. An at least 10 to 1 productivity difference among programmers is common, even on the same project (for instance, Demarco and Lister).[36]

The effect of difference in productivity between *veterans* and *rookies* is shown in Figure 11. Whereas a 25% staffing pulse in this experiment always increases the average production rate (Figure 11(a)), the accumulated production gain for a 30-person team (Figure 11(b)) is sufficient to meet the project target when the difference in productivity between *veterans* and *rookies* is more than 0.8 (curves #4 and 5). For a 40-person team (Figure 11(c)), the project can be completed before the deadline even with a smaller difference in productivity. With higher veterans' productivity, the negative impact of adding *rookies* does not prevent the team from completing the project ahead of schedule.

2.4.2. Effect of team investment in training. In the following experiment it was assumed that the starting team is comprised of 26 *rookies* and four *veterans*, allocated for mentoring 0.25 of their worktime and smaller increments of the *staffing pulse* were applied (2, 4, 6, 8, and 10 *rookies*) (Figure 12). The dynamics of *assimilation rate* during the assimilation of both initial and additional workforces reflect the number of *rookies* (compare Figure 12(a) with the results shown in Figure 8, where all initial personnel are *veterans*). With 0.06 communication entropy in all variants adding manpower delays the project (not shown), whereas with 0.03 communication entropy it diminishingly beneficial (Figure 12(b)).

In ProjScout™, the dynamic impact of training is composed of at least two opposite effects – decreased

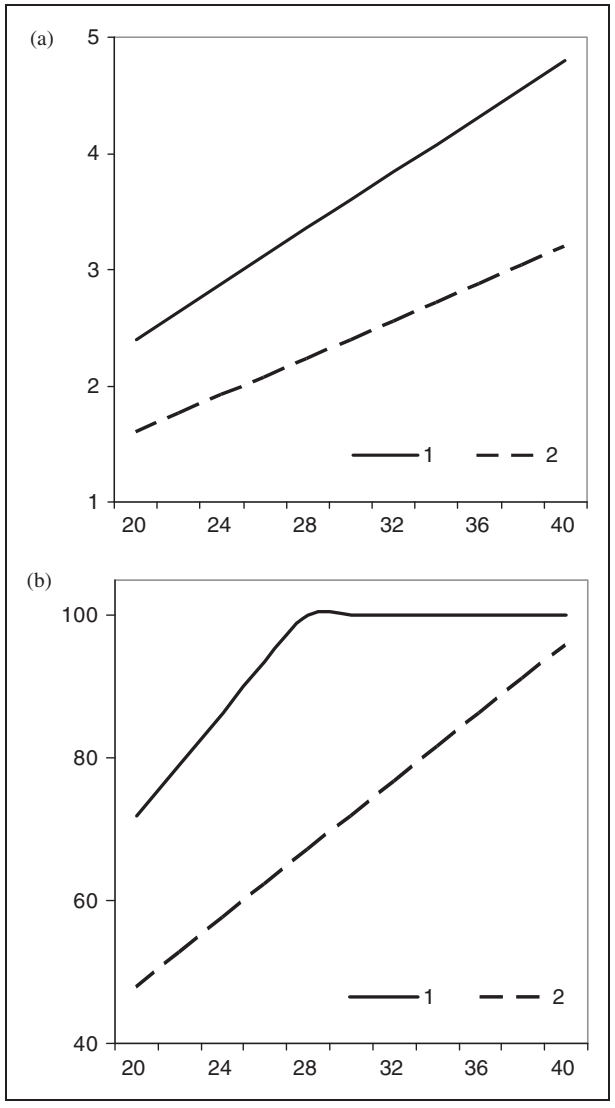

Figure 10. ProjScout™. Effect of team size on production with no communication overhead and team investment in mentoring considered. X-axes – *total personnel* (people). (a) *production rate* (task/team-day). (b) Work *completion (tasks, %)*. 1 - all *veterans*; 2 - all *rookies*.

production rate due to distraction of veterans from production and increased *assimilation rate* of rookies, which speeds up the increase of veterans' stock. The dependence of production on team experience described above indicates that the time required to reach nominal capacity is determined by the *assimilation rate* of rookie conversion to veterans. The more time required for rookies to be assimilated the more time required to complete the project.

The *assimilation rate* depends on the assumed *individual learning time* and the *team investment in mentoring*. While *individual learning time* apparently can be only estimated, the team *investment in mentoring* can be employed arbitrarily through management policy by changing the allowed number of *mentors* per rookie and the fraction of time allocated for mentoring. The number of *mentors* per rookie can be set either by limiting the number of people working together (project sub-groups) i.e. in pair programming, or by assigning particular team members to guide project newcomers. The fraction of time allocated for mentoring can be set through establishing special days allocated for training, or setting the hours when mentors make themselves available for being distracted from production.

Table 2. Effect of team experience on production parameters (*total personnel* = 35)

veterans	*rookies*	*production rate* (task/day)	Work *completion* (%)
2	33	2.88	86.4
4	31	2.96	88.8
6	29	3.04	91.2
8	27	3.12	93.6
10	25	3.20	96.0

Figure 13 shows the effect of investment in mentoring with allocated for mentoring only 1 hr a day. Without investment in mentoring (curve #1), adding five rookies (25% of initial personnel) brings about an 8% *production rate* increase – from 2.13 to 2.30 task/day. With only 2 hours a day invested in mentoring, the same number of added rookies brings about a smaller production gain to 2.26 tasks a day (curve #2), followed by a further gradual increase, reaching 2.43 tasks a day (14%). However, higher investment in mentoring of 4–8 hours a day (curves #3–5) causes an increasing production drop to 2.07, 1.97, and 1.87 task/day, correspondingly, followed by gradual increases in production rate, reaching 2.46 tasks a day in all variants. Thus, increasing the initial workforce by 25% provides up to a 15.5% increase in production rate, regardless of the amount of investment in mentoring.

The effect of time *allocated for mentorin*g was studied within the range 0.125 (1 hr of a mentor's time per day) to 2 workday/day (either two *veterans* mentoring full-time or four *veterans* mentoring half-time). Figure 14 shows the effect of *investment in mentoring* on production with an assumed *individual learning time* of 30 days. An increase of *investment in mentoring* increases the *assimilation rate* ($y = 0.1212x - 0.0021$, not shown) and the corresponding *mentoring effort* ($y = 3.6252x - 0.0014$, not shown). The interaction of both effects – distraction of *veterans* from production and increase of veteran stock during the course of assimilation – results in a non-linear gain of work *completion* (Figure 14(a), $y = 6.2868Ln(x) + 85$). The *production rate* (Figure 14(b)) grows non-linearly with increase of team *investment in mentoring*, but after reaching a maximum

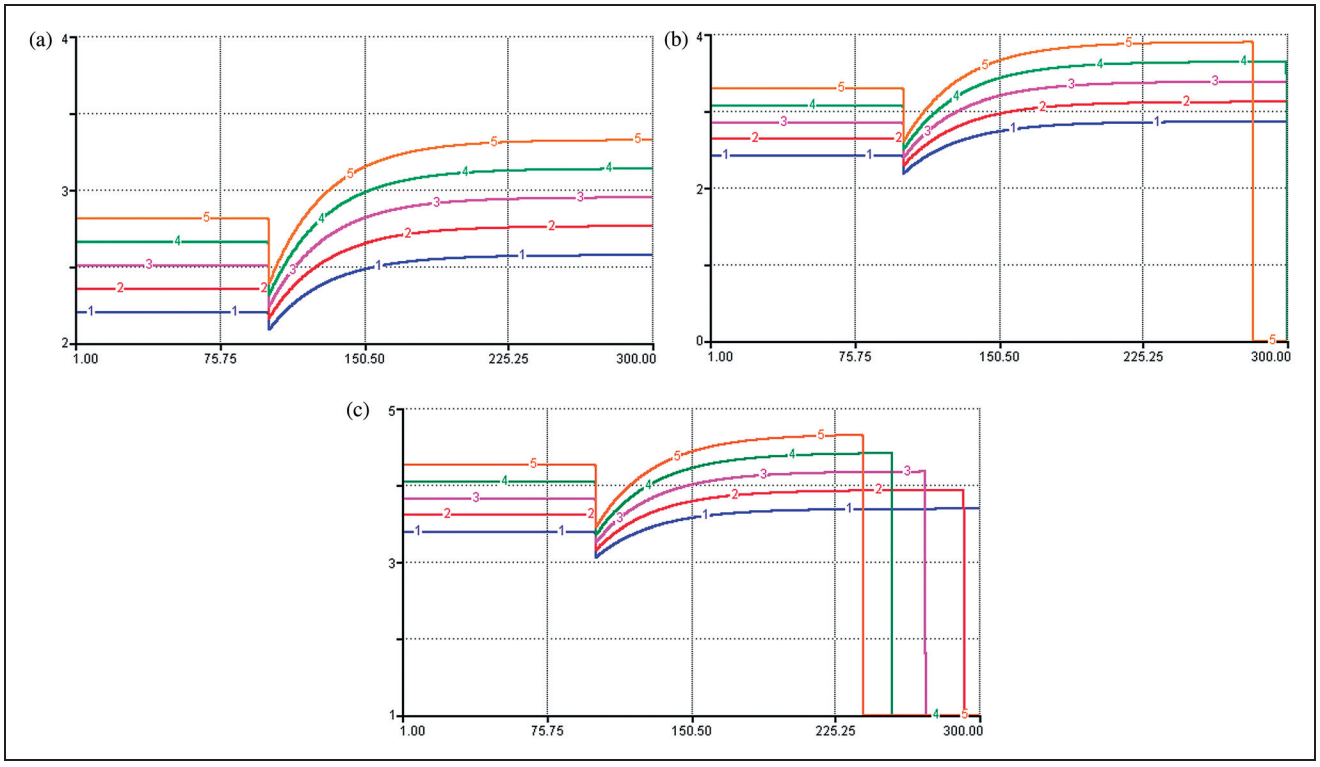

Figure 11. ProjScout™. (a)–(c) Sensitivity of production rate to varying personnel productivity: (a) four to 16; (b) six to 24; (c) eight *rookies* added to 32 *veterans*. 1: 0.9 and 1.1; 2: 0.8 and 1.2; 3: 0.7 and 1.3; 4: 0.6 and 1.4; 5: 0.5 *rookie productivity* and 1.5 *veteran productivity*. *entropy factor* – 0.03, *mentors* – 4, *allocated for mentoring* – 0.25 workday/day.

at 0.75 workday/day stays flat, regardless of further increase. This experiment shows that investment in mentoring may have high and non-proportional effects on the schedule. If an underinvestment in mentoring (1 hr per day) leads to a missed schedule with only 85% of work being completed, the same team will meet the schedule target if 1–2 experienced workers will mentor full-time.

The effect of *individual learning time* was tested at four levels of team *investment in mentoring* for a team of two *veterans* and 26 *rookies*. The *assimilation rate* (Figure 15(a), curves #1–4) negatively responds to *individual learning time* and positively responds to investment in mentoring. With increase of the assumed *individual learning time* from five to 50 days (Figure 15(b)), work *completion* decreases linearly or almost linearly (at 2 workday/day).

The *production rate* (Figure 15(c), curve #1) slows down by the high *individual learning time* only when *investment in mentoring* is low (1 hr/day). With 4 hr a day invested in mentoring (curve #2), the *production rate* only decreases slightly when *individual learning time* exceeds one month. At a high *investment in mentoring* (1–2 workday/day), the *production rate* did not suffer regardless of the *individual learning time* (Figure 15(c), curve #3 (the curve corresponding to 2 workday/day is not sh\aown because of overlap).

2.4.3. Effect of communication overhead. A direct consequence of Brooks' Law is that intercommunication directly conflicts with production. The communication overhead is usually modeled as a product of group communication paths and an empirical 'need to communicate' factor.[27] The sensitivity of the ProjScout™ to the communication *entropy factor* was calibrated at a constant value of *nominal production*.

An initial experiment with *staffing intervention* showed how increased *communication overhead* disproportionately consumed productive time. With nominal values of *entropy factor* and *nominal production* the schedule could not be met, either by increasing initial staff or by staffing corrections, since the communication overhead consumed all productive time added by the increased workforce. The *communication overhead* increases non-linearly with team size (Figure 16(a)). For a 40-person team, it reaches 20% at 0.03 entropy factor (curve #1 $[y = 0.06x2 + 1.05x + 4.6]$ and 40% at 0.06 entropy factor (curve #2 $[y = 0.12x2 + 2.1x + 9.2]$).

At 0.03 entropy factor the *production rate* (Figure 16(b), curve #1) and work *completion* (Figure 16(c), curve #1) increase with team size. At 0.06 *entropy factor* the *production rate* (Figure 16(b), curve #2, $[y = -0.013x2 + 0.2x + 2]$) and work *completion*

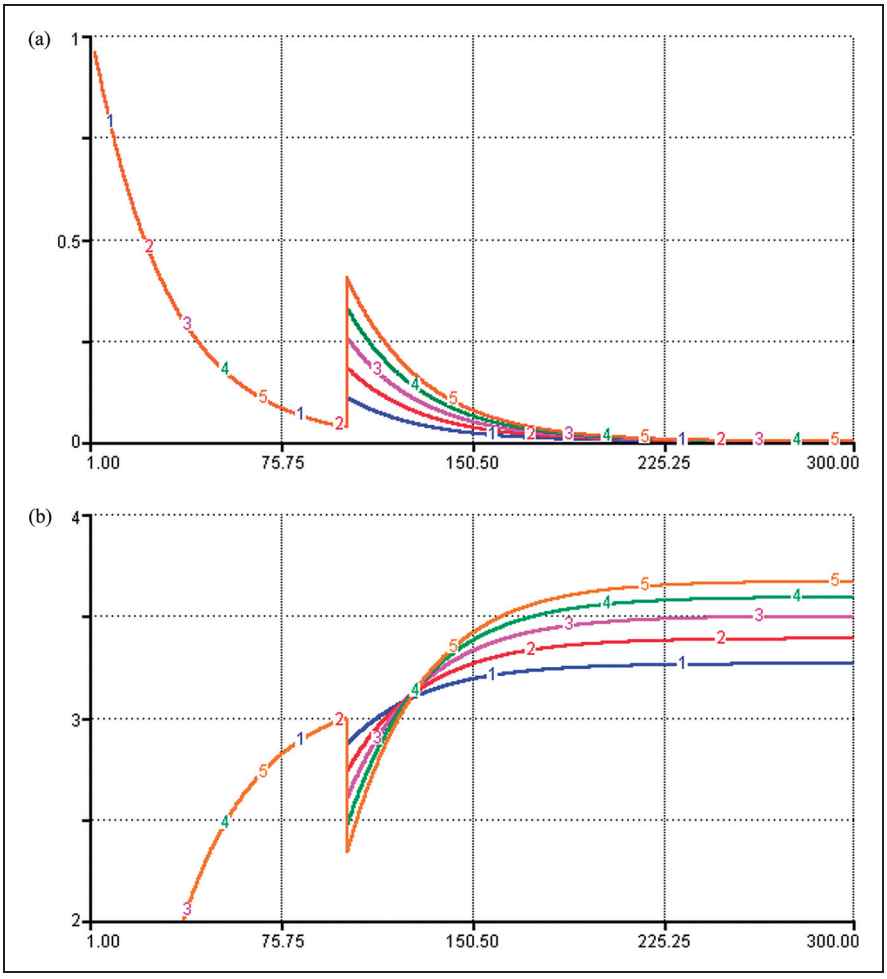

Figure 12. ProjScout™. Sensitivity of mentoring effort to staffing correction of rookie's team. 1: 2; 2: 4; 3: 6; 4: 8; 5: 10 *rookies* added to 30 people-team (four *veterans* and 26 *rookies*). (a) *assimilation rate* (person/day); (b) *production rate* (task/team-day). *entropy factor* – 0.03, *mentors* – 4, *allocated for mentoring* – 0.25 workday/day.

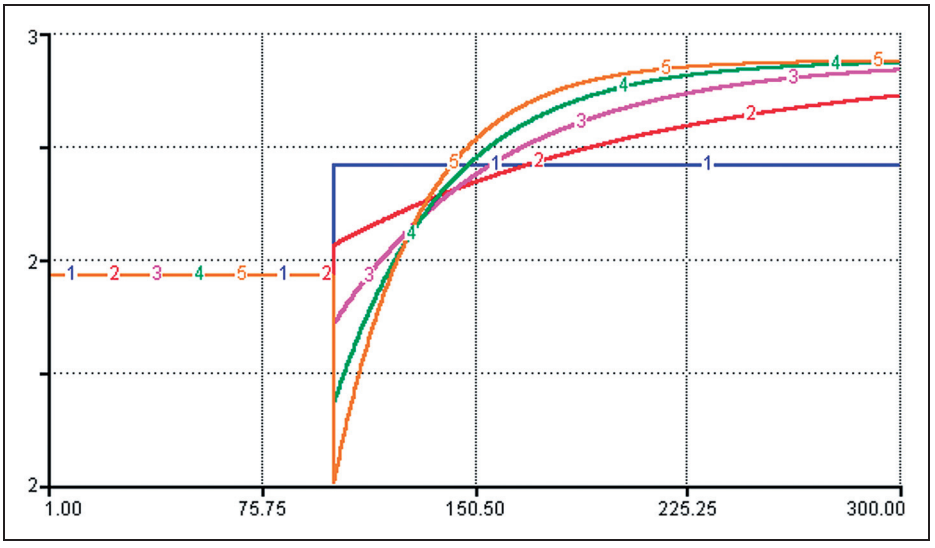

Figure 13. ProjScout™. Sensitivity of *production rate* to varying *investment in mentoring* (person-day). 1: 0; 2: 2; 3: 4; 4: 6; 5: 8 *mentors*. *staffing pulse* – 5 *rookies*, *allocated for mentoring* – 0.125 workday/day. All other parameters are as in Figure 6(a).

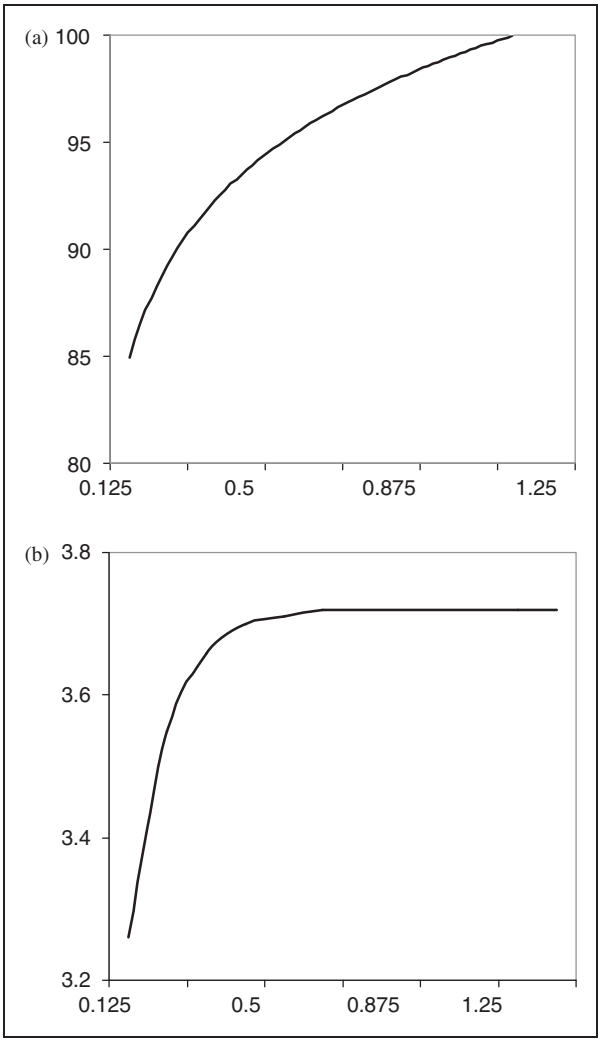

Figure 14. ProjScout™. Effect of investment in mentoring (person-day). (a) Work *completion* (tasks, %). (b) *production rate* (task/team-day). Initial team: two *veterans* and 29 *rookies*. *individual learning time* – 30 days.

(Figure 16(c), curve #2, $[y = -0.39x2 + 6x + 58]$) decline when the team exceeds 32 people. At 0.03 *entropy factor*, the *actual productivity* for a 40-person team is approximately 23% less, and at 0.06 *entropy factor* 46% less, than for a 20-person team (Figure 16(d)).

Stutzke[30] viewed communication overhead as a second-order effect that cannot be quantified directly. There are no specific values for communication entropy or programmer's productivity in TMMM.[11] Within the boundaries of the model, the *communication overhead* that can be tolerated before it will jeopardize the work depends on the *nominal production* and the value of the communication *entropy factor*. With 0.03 communication *entropy factor*, a team of 34 experienced workers can complete the project six days ahead of schedule, and increases in team size result in early project completion.

2.5. 'Real-life' scenario

To facilitate comparison and understanding of the model's behavior in the above tests, it was assumed that the starting team is comprised of either all veterans or all rookies. This is not the case in reality – normally a project starts with a few experienced people who provide initial guidance for new arrivals. For instance, the SEL recommend starting software projects with a small senior staff and adding staff after initial requirements and architecture work is mostly complete.[9] To simulate such a 'real-life' team it was assumed that there were four *veterans* available at the start and 30 days of *individual learning time* for new hires to become productive. Since the actual values of *nominal production*, *veteran productivity*, *rookie productivity*, and *communication entropy* are not yet known, the only parameters available for playing with scenarios are the initial team size, the size of the *staffing pulse*, and the *investment in mentoring*.

A steady-state simulation shows that without taking the *communication overhead* into account, a team of four *veterans* and 28 *rookies* can complete 98.8% of the work before the deadline. Increasing the initial team by just four more *rookies* ensures work completion 25 days ahead of schedule. However, with even moderate communication (0.03 *entropy factor*) and training overheads (0.5 workday/day of *investment in mentoring*), the 32-person team will complete only 84%, and the 36-person team only 90% of the job before the deadline (Figure 17(a)). Staffing intervention does not help much. Adding up to 12 *rookies* to a 32-person team improves work *completion*, but the more people added, the weaker the improvement (curve #1, $y = -0.32x^2 + 2.7x + 84$) (Figure 17(a)). For the 36-person team, adding up to six additional *rookies* improves work completion, but more than nine *rookies* delays production (curve #2, $y = -0.35x^2 + 1.9x + 90$).

Whereas staffing intervention always decreases *actual productivity* (Figure 17(b), curve #1 $[y = -0.0049x + 0.0979]$, curve #2 $[y = -0.0045x + 0.1027]$), the *production rate* depends non-linearly on the initial team size (Figure 17(c), curve #1 $[y = -0.0193x2 + 0.1987x + 3.25$, curve #2 $[y = -0.0229x2 + 0.1671x + 3.466]$), expressing the effects of Brooks' Law. The positive effect of added manpower declines ($y = -0.616x + 2.938$): adding three *rookies* increases work completion by 2%, adding six *rookies* – only by around 1%, and adding 10 *rookies* has no effect. Overall, with *nominal* estimated *production* and communication and training overheads, staffing intervention cannot help to meet the schedule.

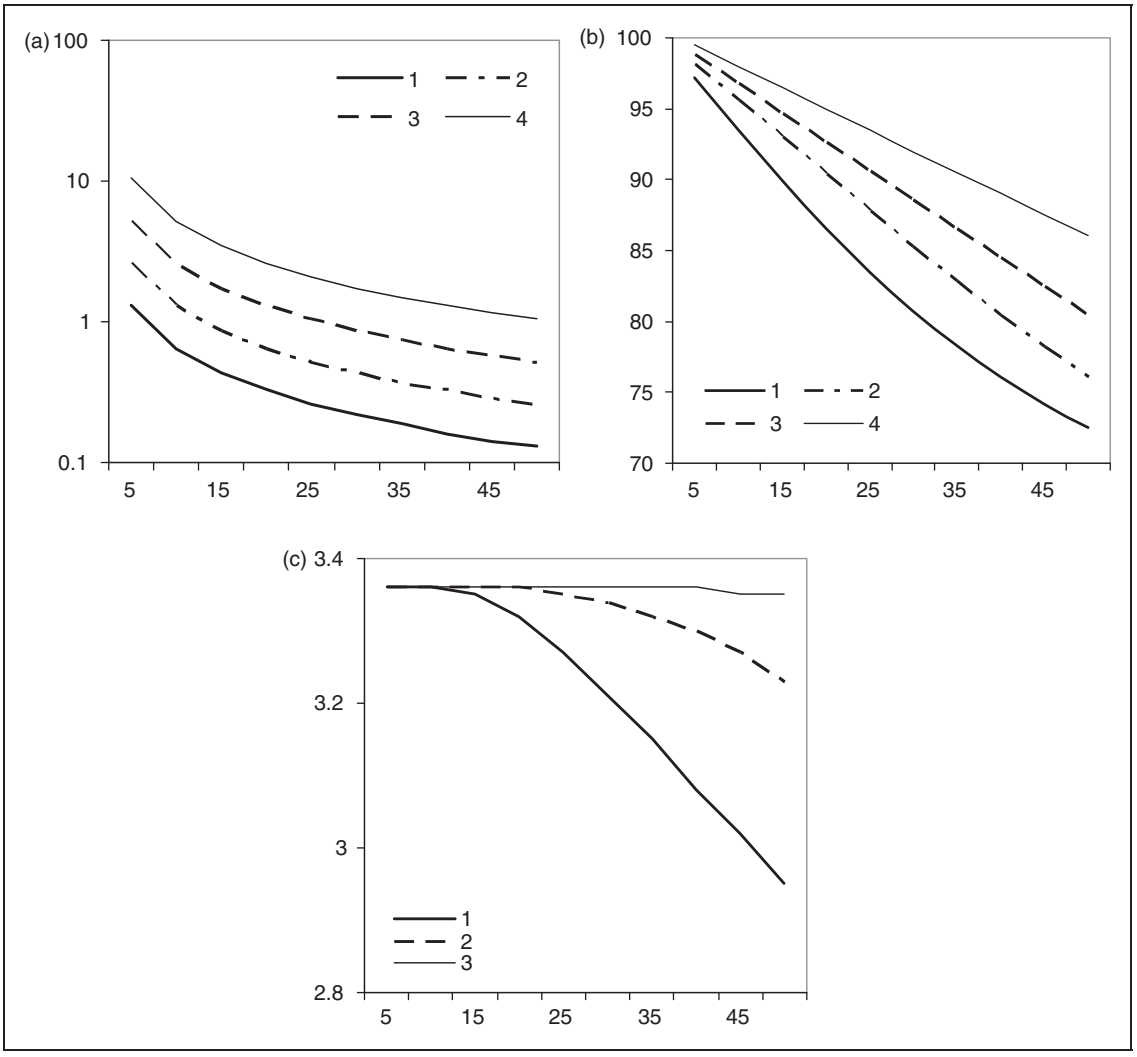

Figure 15. ProjScout™. Effect of individual learning time and investment in mentoring. (a) *assimilation rate* (person/day). (b) Work *completion* (tasks, %). (c) *production rate* (task/team-day). *X*-axes – *individual learning time* (days), *team investment in mentoring*: 1: 0.25; 2: 0.5; 3: 1.0; 4: 2.0 workday/day. Initial team: two *veterans* and 26 *rookies*.

Since production and overhead parameters are either unknown or most probably beyond management control, the only variable left to play with is team *investment in mentoring*. With four *veterans* mentoring full-time, the *assimilation rate* increases from 0.47 to 3.73 persons/day (Table 3). As a result, although no work will be accomplished during the first week of the project, the *investment in mentoring* will eventually pay off. The rate of *production rate* recovery is so high that the initially understaffed 32-person team can meet the schedule (Figure 18). The investment in mentoring can significantly speed up the project only when the difference in productivity between *veterans* and *rookies* is sufficiently high. Even the 32-person team can meet the project target when the *veteran productivity* is above 1.3. In the above experiments we assume the fraction of mentors as constant. The adoption rate can be enhanced if some of the personnel are charged with tutoring when they became *veterans*.

3. Discussion

3.1. Putting things in context

Simulating project scenarios is becoming an increasingly popular way to reduce project risks and to help in staffing decision-making at the earliest stage possible.[2] There are numerous management methods, both proactive, or those used for estimating effort, resources and quality, project sizing, work breakdown structures, scheduling, critical path analysis, risk evaluation, and what-if sensitivity analysis of cost drivers,[1,2,37,38] and reactive, or those used for dealing with the problems that have already arisen. Yet the *practical* assessment of

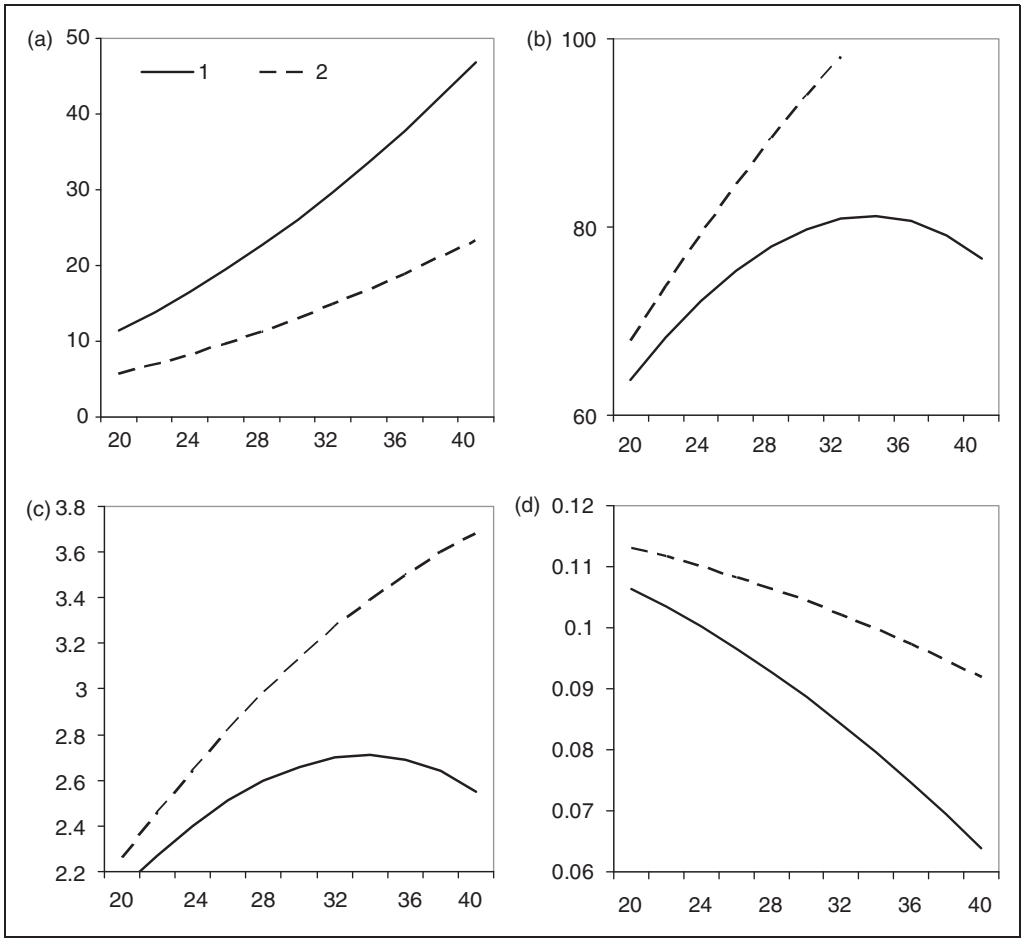

Figure 16. ProjScout™. Effect of communication entropy. *X*-axes – *total personnel* (people). *entropy factor*: 1 – 0.06; 2 – 0.03. (a) *communication overhead* (%). (b) *production rate* (task/team-day). (c) Work *completion* (tasks, %). (d) *actual productivity* (task/person-day).

project risks, cost, and schedule remains more art than science – it is still mainly dependent on a manager's experience, 'hunches', 'gut feelings', and application of informal industry rules of thumb.[36,39] Why?

Based on the notion of Kolmogorov complexity, Lewis[42] showed that a programmer's productivity cannot be objectively determined or compared across projects. While programming time and effort are objectively linked to the algorithmic complexity of the tasks, complexity of the human element involved in a software project makes things nearly unpredictable.[40] To adjust development productivity to fit a larger range of problems, descriptive static third-order models, such as COCOMO II, incorporate a set of environmental factors, but at the cost of a much higher number of inputs. Seven of the 22 COCOMO II cost factors relate to personnel capability. Collectively, they exert an influence range of 25.8 at the base effort estimate. For instance, a project employing analysts from the bottom 15% of the productivity scale will require 1.51 times as much effort to reach completion as one employing analysts from the top 10%.[41] In addition to individual difference in productivity between the most and the least effective developers[36], team cohesion has a greater impact on productivity than the team members' individual capabilities.[43]

No static model, regardless of how comprehensive (hundreds of parameters are required as input in some commercial tools), can account for the dynamic nature of the development process, to predict emergent effects of the second order, such as in Brooks' Law. More recently, researchers are building models of projects that are both systemic and dynamic, and explain many project behaviors that conventional decomposition models do not.[37] However, the inherent complexity of a modeled domain creates a barrier that no level of mathematical technique and no amount of individual expertise has yet been able to overcome.[44] Multiple non-linear relationships and feedbacks result in overwhelmingly complex dynamics of the software project: effects of project size and complexity, requirements creep, quality control, sequential constraints,

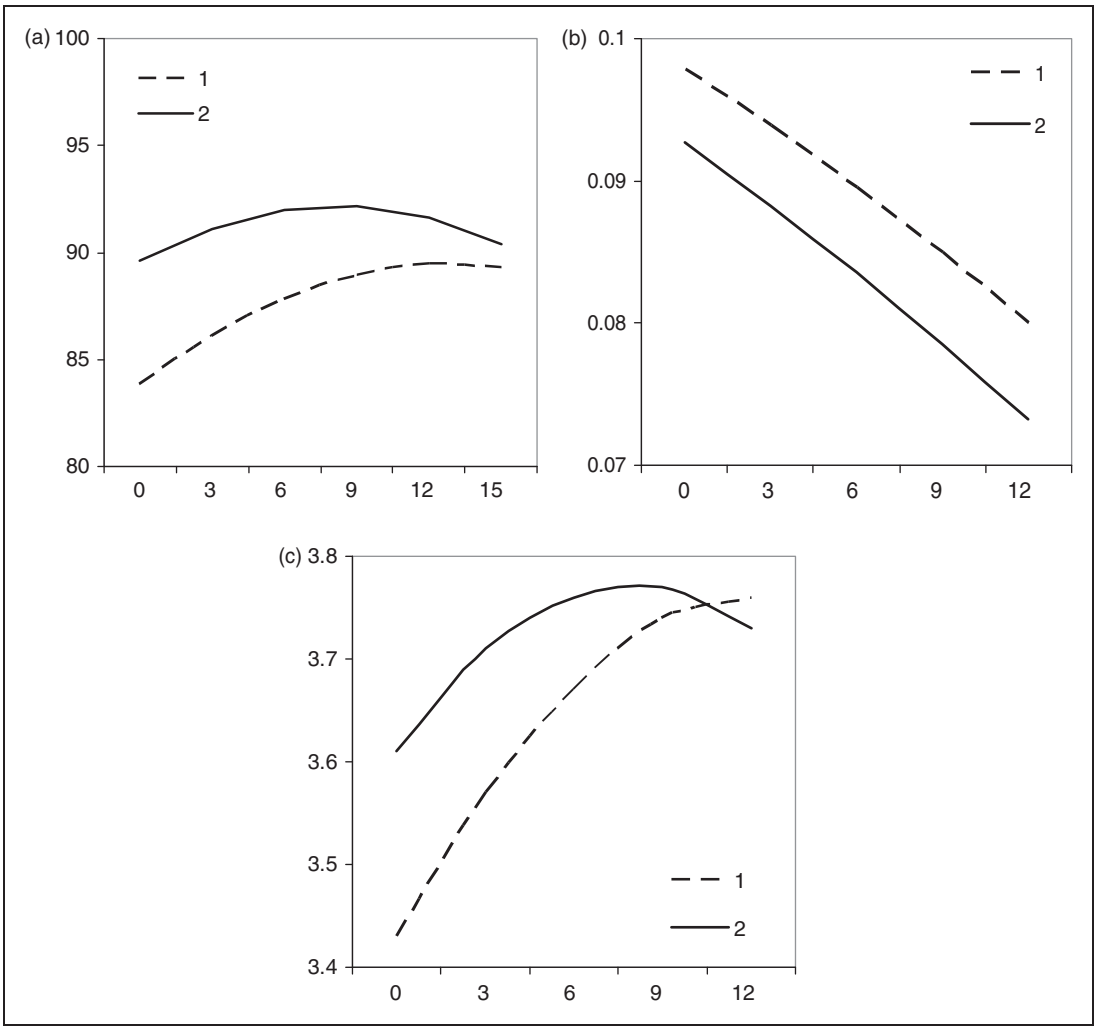

Figure 17. ProjScout™. Effect of initial team size and staffing correction under moderate communication overhead. X-axes – added manpower (*rookies*). (a) *work completion* (tasks, %). (b) *production rate* (task/team-day). (c) *actual productivity* (task/person-day). Initial team – four *veterans* and: 1–28 *rookies*; 2 – 32 *rookies*. *entropy factor* – 0.03; *individual learning time* – 30 days; *investment in mentoring* – 0.5 workday/day.

development tasks division, staffing policies, schedule pressure, staff turnover, skills attrition, team cohesion, communication entropy, organizational learning, on-the-job training, and others. Some empirical dynamic models require hundreds of parameters as input, with a single error capable of having disproportionate impact on results. As the model's complexity increases, the likelihood of error grows. John Sterman[33] states that the models "are often so poorly documented and complex that no one can examine their assumptions They are black boxes. They are so complicated that the user has no confidence in the consistency or correctness of the assumptions. They are unable to deal with relationships and factors that are difficult to quantify, for which numerical data do not exist, or that lie outside the expertise of the specialists who built the model . Because computer models are so poorly understood by most people, it is easy for them to be misused, accidentally or intentionally. Thus, there have been many cases in which computer models have been used to justify decisions already made and actions already taken, to provide a scapegoat when a forecast turned out wrong, or to lend specious authority to an argument."[33]

Table 3. Effect of *investment in mentoring* on *mentoring effort* and *assimilation rate* (veterans = 4, rookies 28)

investment in mentoring (workday/day)	*mentoring effort* (person-days)	*assimilation rate* (person/day)
0.5	14	0.47
1.0	28	0.93
4.0	112	3.73

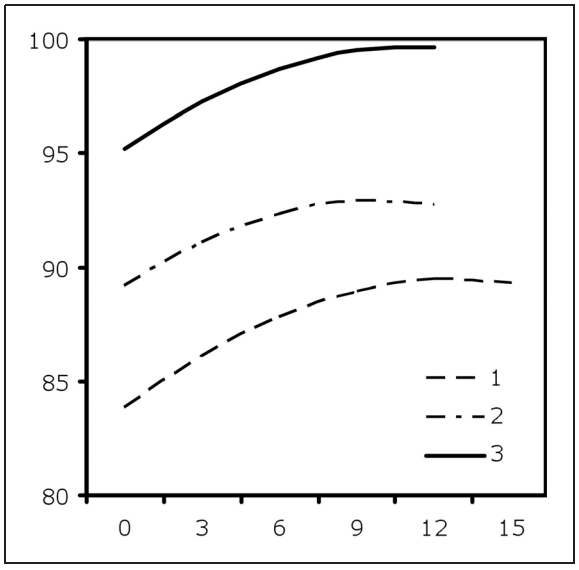

Figure 18. ProjScout™. Effect of investment in mentoring on work *completion*. X-axes – added manpower (*rookies*). 1: 0.5 workday/day; 2: 1.0 workday/day; 3: 4.0 workday/day. Initial team: four *veterans* and 28 *rookies*, *entropy factor* – 0.03, *individual learning time* – 30 days.

The policy recommendations suggested by dynamic models can be counterintuitive. For instance, the interaction of resource allocation delays and various controls imposed by managers suggests counterintuitive project control recommendations, i.e. exercising less control to shorten project durations.[23] What do we do if the simulation experiments produce results that do not confirm the theory prediction? One of the two must be wrong, either the model or the theory. One approach to this dilemma is expressed in the following: ''This exercise has demonstrated that a small-scale and simple model can help shed light on process phenomena. Even though the model contains fairly simple formulations, the dynamic time trends would be very time consuming for a person to manually calculate and all but impossible to mentally figure.''[29] Another approach humbly acknowledges, ''All models are wrong. We must remain mindful of the fact that we cannot establish truth through System Dynamics modeling We can never place our mental models on a solid foundation of Truth because a model is a simplification, an abstraction, a selection.''[47] A model, whether static or dynamic, compromises between completeness for validity and simplicity for communication. A model is only a modeler's construct, a combination of exact equations, assumptions, and arbitrary values.[6] An oversimplification may undermine validity. Both oversimplification and high complexity effectively prevent models from being considered by practitioners: ''much of the mathematical modeling, producing ever more complex solutions to ever more complex models, is motivated by mathematical impressiveness rather than the need to solve real world problems''.[36]

Where is the golden mean between validity and clarity? The toughest problem a modeler faces is ''seeing'' that a problem exists, discerning or conjecturing the relevant variables, and guessing the relations among them'.[45] Rutherford Aris[45] suggests addressing this problem by starting with the so-called conservation principles (or balance laws), whether well corroborated or just guessed. The conservation principle is a generalization capturing the 'larger picture' of reality, a generalization that derived from extensive empirical evidence, a theory that we expect to hold regardless of input fluctuations. The conservation law cannot be overly limiting: it must generalize; or it is not a law. While a counterintuitive simulation output can sometimes be explained by the emergent dynamic interaction of other effects, the overall behavior of a model should be consistent with the relevant descriptive knowledge. The conservation principle, which holds regardless of input fluctuations, allows prediction of the macroscopic behavior of a system without analyzing the microscopic details of its course. Compliance with the law should validate a model, not vice versa. A model's failure to correspond to the law's prediction most probably indicates a problem with the model. In such a case, the job of the modeler is to improve the model until it represents the law sufficiently well, thus becoming useful.[46]

As pointed out by Robert Aumann,[46] we cannot expect game and economic theory to be descriptive in the same sense that physics or astronomy is. The discovery of a conservation principle is an extraordinary achievement in any field of scientific endeavor, and is particularly difficult in a computationally irreducible domain, 'congested' as this is with uncertainties and soft factors.[17,19] Fred Brooks described a conservation principle in a computationally irreducible domain that makes this achievement particularly remarkable. While being 'outrageously oversimplifying', Brooks' Law holds as' the best zeroth-order approximation to the truth, a rule of thumb to warn managers against blindly making the instinctive fix to a late project'.[5] Like many other intellectual breakthroughs, the paradoxical, unintended effect of added manpower revealed by Brooks' Law became 'apparent' only once it was expressed in a catchy, proverbial form. Due to constant challenge by numerous empirical and modeling studies, Brooks' Law became one of the best-defined archetypal cases of a systemic problem. Emphasizing the systemic nature of software projects, Brooks points out the type of problems that cannot be solved by a straightforward addition of resources. The unintended consequences[6] are an increased loss of productive time due to

communication overhead and training effort, and a loss of productivity due to burnout and skill attrition from increased turnover.[7,8] Lessons learned from dealing with Brooks' Law convey the wisdom that may help us recognize 'the things we can change' in dealing with other systemic, seemingly unsolvable problems.

Over the years, an interest in TMMM spread from software engineering and computer science to management and information systems. There are references to TMMM in 50 different disciplines, including business, management, ergonomics, engineering, operations research, law, planning and development, psychology, economics, social issues, chemistry, oncology, physics, anatomy, physiology, energy and fuels, and more.[17] A content analysis of 497 articles citing TMMM over the period 1975-1999 shows that almost a quarter of 574 citation contexts point to 'Brooks' Law', and its prominence holds for most of the higher-ranking context subject areas, including software engineering, computer science, management, and other science and information systems.[19] For instance, in the case of sales force according to Varjan,[48] ''adding people leads to lost effort through training, communication, administration, and other overhead. A rough rule is that for every person you add to a project team, you have to subtract 10% accumulative from that person's effort. Adding a new person thus results in 90% additional effort and adding a second person results in that person adding only 80% effort, and so on''.

While a complete account of the aptness of attributing all the citations found by content analysis to Brooks' Law is hardly possible, its ubiquity alone suggests that the term itself may occasionally be used rather in a metaphoric sense. Indiscriminate use of the 'management gospel' may lead to disappointment. If we ask a wrong question, we most probably will get a wrong answer. The inherent negativity of Brooks' Law deters its acceptance, which is further complicated by the lack of clear understanding its applicability. The conclusion that added workforce increases project costs[27,30c] seems to be obvious. However, any cost assessment must include the price of schedule extension, and direct and indirect penalties in case of late delivery, which are not included in the model.[27] Even as a rule of thumb, Brooks' Law applies only when project resources are limited. It is not the case in 'bazaar', open-source development,[10] where production and training processes are not competing for the same resources and the increase in communication does not represent added work.

3.2. Communication: overhead and effort

Estimation tools, such as ESTIMATE PRO and DOORS, account for communication as 'communication overhead'. The Tassc Estimator considers the effect of intercommunication as a noise factor.[49] Abdel-Hamid and Madnik[27] define communication overhead as the average team member's drop in productivity below his nominal productivity, a result of team communication including verbal communication, documentation, and any additional work, such as that due to interfaces. It is widely held that communication overhead increases in proportion to n^2, where n is the size of the team. Communication overhead is zero when the software is developed by one person, and for $n = 30$, it is approximately 50%.[27] If this is the case, should management try to reduce team communication in order to increase the average productivity? The effect of Brooks' Law should not be confused with large organization entropy, accounted for as a B-factor.[50] It is principally important to distinguish between communication overhead (synonyms: scale or communication entropy, 'noise factor', downtime, communication penalty) and the work-related communication effort, which must be accounted separately. Interpersonal communication (task clarification, prioritization, requirement negotiation, informal information exchange, and more) is an important part of the work routine. According to a classic IBM study, software engineers spend only 30% of their time working alone. Half of their working time is spent in groups of two to three people, and the remaining 20% in larger groups and travelling.[51] Software engineers participate in meetings, discussions, training sessions, and other types of social interaction.[36] A more recent study shows that the engineers of a telecommunication company spent more than 6.5% of their time at registered meetings, and 88% of the meetings were project related: technical discussions, 30%; project meetings, 28%; planning meetings, 16%; reviews, such as code inspections and the like, 10%; group meetings, 9%; kick-off meetings (project initiation), 4%; and education and training, 3%.[52] Managers, analysts, and architects spend much more time on work-related communication. Non-explicit communication effort may constitute a major share of all project efforts. While Brooks' Law may apply to workalone activities, such as coding, systems integration, quality control, documentation, and others, it does not apply at algorithm development stages, or whenever a significant research and development (R&D) effort is needed. Brooks' Law does not apply when the informal communication is the work. It was shown that smaller teams and agile development models minimize the negative impact of organization entropy. The case study by Menlo Innovations describes how strictly following extreme programming (XP) practices may overcome Brooks' Law; creating on-the-job training and constant communication

within the team enabled new developers to learn faster and contribute to the team productively within weeks rather than months.[53] Sharpe[54] describes the unique working environment at Menlo Innovations: 'All developers are co-located in the same large room (no offices or cubes) and pair program 100% of the time – they follow strict XP practices. A scheduling team determines which projects developers work on and whom they pair with on a weekly basis. So developers work with different team members and possibly different projects every week'.

Although not a universal law of software development, Brooks' Law is a universal law in the software industry. Since 'bazaar', open-source development can hardly be considered managed; loss of effort due to the duplication of effort and lack of coordination is unavoidable. The way to speed up the FOSS project may require better coordination of effort, and that, in turn, would require greater communication. Theoretically, if development tasks can be ideally partitioned, the production rate may increase proportionally to the available workforce. Moreover, the increase in the number of communication paths may speed up development by turning it into a kind of brainstorming session, facilitated by instant messaging, document sharing, and teleconferencing. Nevertheless, the increase in the number of communication paths does not affect the production rate because communication effort does not turn into overhead. Brooks' and Linus' laws are not competing, but rather mutually complimentary theories, addressing respectively different, 'cathedral' and 'bazaar', development models. The effect of intercommunication on production rate through increasing team cohesion should be reflected in the model of cathedral development. The specific, empirical value of the 'need to communicate', expressed in both communication overhead and work-related communication effort, depends on project type and complexity, organization policy, diversity, and physical proximity of team members, social interaction (the 'coffee machine' effect), team cohesion, and other dynamic factors of the second order.

3.3. Training

Michael Polanyi[55] defines personal knowledge, as ''An art, which cannot be specified in detail, cannot be transmitted by prescription, since no prescription for it exists. It can be passed on only by example from master to apprentice. This restricts the range of diffusion to that of personal contacts . By watching the master and emulating his efforts in the presence of his example, the apprentice unconsciously picks up the rules of the art, including those, which are not explicitly known to the master himself''. The specific suggestions for enhancing knowledge management in software companies include on-the-job training to reduce dependency on a few knowledgeable people, encouraging developers to document and store their experience, sharing knowledge through meetings rather than through documents, and the like.[56] On-the-job training is increasingly a part of the work routine for both rookies and experienced performers. Ryan and O'Connor[57] found that while efficiency, i.e. 'doing things right' in software development teams is associated with explicit job knowledge (familiarity with written procedures), expertise, formal knowledge sharing, and administrative coordination, effectiveness, i.e. 'doing the right thing' is based on tacit knowledge and non-formal procedures. This tacit knowledge becomes externalized through iterative, face-to-face interaction.[58] Tacit knowledge may be shared in a number of ways, including mentoring and apprenticeships, but usually involves social interaction. Ryan and O'Connor[57] demonstrated that tacit knowledge is related to the quality of social interaction. There is a need for knowledge sharing to enable software organizations to leverage tacit knowledge sharing.[58] Team tacit knowledge and the coordination of specialized knowledge within the team are significant factors in effective performance of a software development team. Managers of software development teams can make changes within the organization and in the team to enhance social interaction and encourage the sharing of tacit knowledge, thus increasing team effectiveness.[59] Our simulation experiments indicate that the project team resilience can be improved by increasing the investment into training new team members.

4. Conclusions and further research

Contrary to earlier modeling efforts, our purpose was neither to clarify, nor to confirm, Brooks' Law, but was limited to provide a sufficiently transparent, yet robust, representation of the causal relationships among processes specified in TMMM, using Brooks' Law as a principal conservative theorem for team knowledge-dependent production. Rather than trying to challenge Brooks' Law, the compliance was set as a precondition of the model's validity. Brooks' Law requires neither advocacy nor confirmation through simulation experiments. Initial tests performed according to the stated purpose showed that ProjScout^{TM} successfully emulates the effects of Brooks' Law, responding logically when input parameters are assigned extreme values. Further evaluation of the conceptual and operational validity of ProjScout^{TM} found it to be highly and coherently sensitive to stepwise changes of input. The compliance of a theoretical model to the conservation

law under a wide range of conditions essentially validates the model.

However, even if ProjScout™ accurately emulates tradeoffs between production and training and communication overheads, how helpful is it? Can such a simplistic model be used for testing staffing and intervention scenarios? While not a comprehensive project model (no work sequential constraints or team dynamics were included), experimenting with the basic ProjScout™ model provided important insights on available management options. In particular, simulation experiments revealed relative inherent limits of staffing intervention, related to dynamic relationships between initial team size, training new team members, their learning curve, and their productive time lost due to intercommunication. Lowering the communication overhead, by reducing the size of work groups and implementing internal policies that limit communication, cannot be a universal recipe for raising production, because it does not take the necessity of work-related communication into account. While the increase in nominal production or the number of experienced personnel may be beyond a manager's control (the things they cannot change), some other steps having a significant impact on the project's outcome may be tried through management policies. For instance, instead of reinforcing unintended consequences by adding manpower, having people work overtime, or renegotiating the schedule or the entire project, the schedule can be mitigated by increasing the investment in mentoring – even at the expense of a temporary postponement of production. However, the actual effect of investment in mentoring, as well as concluding whether Brooks' Law applies in a particular case, can be modeled only when actual values of the productivity, communication entropy, and actual learning curve are known. Since all parameters of the ProjScout™ model are arbitrary, its real value is its' robustness and ability to withstand calibration using empirical parameters.

The ability to accommodate dynamic and emergent effects of the second order could make existing parametric models more helpful for project planning and real-time decision support. Well-understood and robust dynamic models may serve as the building blocks for more valid project models, encapsulating characteristic process patterns and industry rules of thumb, and thus becoming suitable for simulating project scenarios. Choi and Bae[60] successfully combined the static estimation models of COCOMO II with system dynamics, and applied an expert judgment technique, Delphi, to overcome the limitations of project data. A formal model that makes the conservation principle a part of the estimation algorithm, may empower the manager with the ability to experiment with different scenarios. After establishing a reasonable range of numbers using first-order approximations, the manager can use the comprehensive parametric tool with built-in conservation principle to quantify the estimation according to the project details and organizational realities.

To develop into a more complete project model, ProjScout™ needs to accommodate cost factors from comprehensive parametric models and dynamic effects of the second order (work sequential constraints, requirements creep, personnel turnover, team cohesion, learning curve decay, project type-dependent communication effort, organization overhead, etc.). Fortunately, the behavioral robustness and structural clarity of the basic ProjScout™ enables one to expand the boundary of the model without running into overwhelming complexity. ProjScout™ can be further localized to facilitate decision-making in managing human capital in a wide range of production and service industries, for instance, in sales representative or resident physician training – wherever experienced performers have to divide their productive work time between actual production and training co-workers, group communication, coordination, and other activities.

Acknowledgments

Many thanks go to Professor Marvin Edelman at the Weizmann Institute of Science for editing the manuscript, Ms Sarah Davie at ISEE Systems, Inc., for correcting the formulation of the staffing rule, Ms Dvorah Olam at the Hebrew University-Hadassah Medical Center for proofreading, Ms Shirelle Dashevsky for continuous support and encouragement, and the reviewers for comments and suggestions.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

1. US Air Force's Software Technology Support Center. 'Guidelines for Successful Acquisition and Management of Software Intensive Systems (GSAM): Weapon Systems, Command and Control Systems, Management Information Systems. Part 1: Acquisition. Chapter 2. Software Victory - Exception or Rule?', Version 3.0, 2000 May [cited 2005 Aug 12]. Available from: http://www.stsc.hill.af.mil/resources/tech_docs/gsam3.html.
2. Institution of Civil Engineers and the Faculty and Institute of Actuaries. *RAMP* – Risk Analysis & Management for Projects – A Strategic Framework for Managing Project Risk and its Financial Implications. Thomas Telford, Ltd. London.
3. NASA. *Lack of disciplined cost-estimating processes hinders effective program management*. United States General Accounting Office. Report No.: GAO-04-642. May, 2004 [cited August 12, 2006]. Available from: http://www.gao.gov/new.items/d04642.pdf.

4. Standish Group International. *The chaos report*. V. 3. 2003.
5. Brooks FP, Jr. No silver bullet – essences and accidents of software engineering. *Computer* 1987; 20: 10–19.
6. Wolstenholme EF. Towards the definition and use of a core set of archetypal structures in system dynamics. *Syst Dynam Rev* 2003; 19: 7–26.
7. Oliva R and Sterman JD. Cutting corners and working overtime: quality erosion in the service industry. *Manage Sci* 2001; 47: 894–914.
8. Sterman JD. *Business dynamics: systems thinking and modeling for a complex world*. USA: Irwin/McGraw-Hill, 2000.
9. McConnell SC. Brooks' Law repealed. *IEEE Softw* 1999; 16: 6–8.
10. Raymond ES. The cathedral & the bazaar: musings on Linux and open source by an accidental revolutionary. O'Reilly Media, Inc. Sebastopol CA 95472, 2001.
11. Brooks FP, Jr. The mythical man-month: essays on software engineering. *20th Anniversary Edition*. Sydney: Addison-Wesley, 1995.
12. Weinberg GM. Quality software management: system thinking. Vol. I. Dorset House Publ., New York. 1992.
13. Hsia P, Hsu C and Kung DC. Brooks' Law revisited: a system dynamics approach. In: *Proceedings of the International Computer Software and Applications Conference (COMPSAC '99)*, 1999, pp.370–375.
14. Blackburn J, Lapre MA and Van Wassenhove LN. 'Brooks' Law Revisited: Improving Software Productivity by Managing Complexity, pp.1–24, August 2006 [cited June 18, 2009]. Available at: http://papers.ssrn.com/sol3/papers.cfm?abstract_id = 922768.
15. Hayes L. 'Development Study: Haste Makes Waste', Computerworld, September 23, 2005 [cited August 12, 2006]. Available from: http://www.computerworld.com/s/article/104846/Development_Study_Haste_Makes_Waste.
16. Nishimura KG and Kurokawa F. *Total factor productivity in Japanese information service industries: firm-level analysis*. 21COE, University of Tokyo MMRC Discussion Paper No 11, 2004.
17. Verner JM, Overmyer SP and Mccain KW. In the 25 years since the mythical man-month what have we learned about project management? *Inform Softw Tech* 1999; 41: 1021–1026.
18. Cole A. Runaway projects: cause and effects. *Softw World* 1995; 26: 3–5.
19. McCain KW and Salvucci LJ. How influential is Brooks' Law? A longitudinal citation context analysis of Frederick Brooks' the mythical man-month. *J Inform Sci* 2006; 32: 277.
20. Gruber J. 'More Aperture Dirt', May 4, 2006 [cited June 18, 2009]. Available from: http://daringfireball.net/2006/05/more_aperture_dirt.
21. Yang Y, Chen Z, Valerdi R and Boehm B. 'Effect of Schedule Compression on Project Effort', 2005 [cited June 18, 2009]. Available from: http://sunset.usc.edu/csse/TECHRPTS/2005/usccse2005-520/usccse2005-520.pdf.
22. Lee Z, Ford DN and Joglekar N. Resource allocation policy design for reduced project duration: a systems modeling approach. *Syst Res Behav Sci* 2007; 24: 1–15.
23. McConnell SC. *Software estimation: demystifying the black art*. Microsoft Press, Redmond, Washington. 2006, p.308.
24. Schweik CM, English R, Kitsing M and Haire S. Brooks' versus Linus' law: an empirical test of open source projects. In: *Proceedings of 9th International Digital Government Research Conference*, 2008.
25. Cox B. What if there is a silver bullet and the competition gets it first? *J Object-Oriented Programming*, June, 1992 [cited June 18, 2009]. Available from: http://www.virtualschool.edu/cox/pub/92JOOPWhatIfSilverBullet/index.html.
26. Drucker PF. *The essential Drucker: the best of sixty years of Peter Drucker's essential writings on management*. Collins Business Essentials: Harper Paperbacks, New York, 2008, p.368.
27. Abdel-Hamid T and Madnick S. *Software project dynamics: an integrated approach*. Inc. Upper Saddle River, NJ: Prentice Hall, 1991.
28. Caulfield CW and Maj SP. A case for system dynamics. *IEEE International Conference on Systems, Man, and Cybernetics* 2001; 5: 2793–2798.
29. Madachy RJ. *Software process dynamics*. IEEE Press, John Wiley & Sons, Inc., Hoboken, NJ, 2008.
30. Stutzke RD. A mathematical expression of Brooks' Law. In: *Proceedings of the 9th International Forum on COCOMO and cost modeling*, Los Angeles, CA, 1994.

30a. Madachy RJ and Boehm BW. Software process dynamics. IEEE Computer Society Press. Early Draft Version 4/99. April, 1999 [cited August 12, 2006]. Available from: http://sunset.usc.edu/classes/cs599_99/spd/spd.html.

30b. Madachy RJ and Tarbet D. Software process improvement and practice. *Softw. Process Improve. Pract*. 2000; 5: 133–146.

30c. Wu M and Yan H. Simulation in Software Engineering with System Dynamics: A Case Study. *Journal of Software* 2009; 4(10): 1127–1135.

31. Williams L, Shukla A and Antón AI. An initial exploration of the relationship between pair programming and Brooks' Law. In: *Proceedings of the Agile Development Conference (ADC'04)*, Salt Lake City, Utah, 2004.
32. Forrester JW and Senge PM. Tests for building confidence in system dynamics models. *TIME Stud Manage Sci* 1980; 14: 209–228.
33. Sterman JD. A skeptic's guide to computer models. In: Grant L (ed.) *Foresight and national decisions*. Lanham, MD: University Press of America, 1988, pp.133–169.
34. Forrester JW. The model versus modeling process. *Syst Dynam Rev* 1971; 1: 133–134.
35. Barlas Y. Formal aspects of model validity and validation in system dynamics. *Syst Dynam Rev* 1996; 12: 183–210.
36. Demarco T and Lister T. *Peopleware: productive projects and teams*, 2nd edn. New York: Dorset House Publishing, 1999.

37. Williams T. The contribution of mathematical modelling to the practice of project management. *IMA J Manage Math* 2003; 14: 3–30.
38. Jones C. Software project management practices: failure versus success. *Crosstalk J Defense Softw Eng* 2004; 17: 5–9.
39. Moløkken-Østvold KJ, Jørgensen M, Tanilkan SS, Gallis H, Lien AC and Hove SE. A survey on software estimation in the Norwegian industry. In: *Proceedings of the 10th International Software Metrics Symposium (METRICS 2004)*, IEEE Computer Society Press, Chicago, IL, 2004, pp.208–219.
40. Connell C. 'Are There Limits to Software Estimation?', January, 2002 [cited August 12, 2006]. Available from: http://www.developer.com/xml/article.php/949921/Are-There-Limits-to-Software-Estimation.htm.
41. Boehm BW, Abts C, Clark B and Devnani-Chulani S. *COCOMO II model definition manual*. Los Angeles, CA: Center for Software Engineering, University of Southern California, 1997.
42. Lewis JP. Large limits to software estimation. *ACM Softw Eng Not* 2001; 26: 54–59.
43. Lakhanpal B. Understanding the factors influencing the performance of software development groups: an exploratory group-level analysis. *Inform Softw Tech* 1993; 35: 468–471.
44. Glass RL. *Software runaways: lessons learned from massive software project failures*. Upper Saddle River, NJ: Prentice Hall PTR, 1998.
45. Bunge M. *Appl Math Model* 1979; 3: 79. Review of Aris R. Mathematical modelling techniques: research notes in mathematics. London: Pitman, 1978, pp.1–191.
46. Aumann RJ. What is game theory trying to accomplish?. In: Arrow K and Honkapohja S (eds) *Frontiers of economics*. Oxford: Basil Blackwell, 1985, pp.28–76.
47. Sterman JD. All models are wrong: reflections on becoming a systems scientist. *Syst Dynam Rev* 2002; 18: 501–531.
48. Varjan T. 'Tomicide Solutions January 2009: Using Salesmanship in Print for Better Sales Force Management and Improved Sales Force Effectiveness', January, 2009 [cited August 18, 2009]. Available from: http://www.varjan.com/articles/0901-january-09-hiring-copywriter-or-expanding-sales-force.shtml.
49. Adens G. *Tassc: estimator technical briefing interpreting and calibrating metrics*. Tassc Ltd, 2002 [cited August 12, 2006]. Available from: http://www.tassc-solutions.com/downloads/Interpreting%20and%20Calibrating%20Metrics.pdf.
50. Jensen RW, Putnam LH Sr and Roetzheim W. Software estimating models: three viewpoints. *J Defense Softw Eng*, February 2006 [cited August 12, 2006]. Available from: http://www.stsc.hill.af.mil/crosstalk/2006/02/0602JensenPutnamRoetzheim.html.
51. Mccue GM. IBM's Santa Teresa laboratory — architectural design for program development. *IBM Syst J* 1978; 17: 4–25.
52. Ochs M and Van Solingen R. Making meetings work. *J Defense Softw Eng*, February 2004 [cited June 18, 2009]. Available from: http://www.stsc.hill.af.mil/crosstalk/2004/02/0402Ochs.html.
53. Opelt K. Overcoming Brooks' Law. In: *Proceedings of the AGILE Conference*, Agile 2008, pp.208–211.
54. Sharpe R. 'Breaking Brooks's Law', Posted in Agile, Enerjy TV, general, innovation, process improvement, software quality, October 7, 2008 [cited August 18, 2009]. Available from: http://www.enerjy.com/blog/?p=244.
55. Polanyi M. *Personal knowledge*: Towards a Post-Critical Philosophy. University of Chicago Press Chicago, Ill, 1962.
56. Nielsen PA and Dolog P. 'State-of-the-Art on Software Project Management Knowledge', p.1–13, 2008 [cited June 18, 2009]. Available from: http://vbn.aau.dk/da/publications/stateoftheart-on-software-project-management-knowledge_91bb6da0-c1f9-11dd-9512-000ea68e967b.html.
57. Ryan S and O'Connor RV. Development of a team measure for tacit knowledge in software development teams. *J Syst Softw* 2009; 82: 229–240.
58. Nonaka I and Takeuchi H. *The knowledge creating company*. New York: Oxford University Press, 1995.
59. Chau T and Maurer F. Knowledge sharing in agile software teams. Logic versus Approximation. Lecture Notes in Computer Science. Springer. 2004; 3075: 173–183.
60. Choi KS and Bae DH. Dynamic project performance estimation by combining static estimation models with system dynamics. *Inform Softw Technol* 2009; 51: 162–172.

Dr David Chernoguz is currently a researcher at the Hebrew University-Hadassah Medical Center, Department of Neurology (Jerusalem, Israel).