Sociology in Software Engineering

Craig Caulfield, Gurpreet Kohli , S. Paul Maj

Edith Cowan University, Perth, Western Australia

Introduction

The sociology of software project management is an often under-represented component in the education and professional development of software engineers even though factors such as team formation, role assignment, motivation, training, hiring, and many other peopleware[18] practices have been identified many times as at least equally important to the success of software projects as the technical[14,16,18,42,44,45,46]. The reasons for this may be two-fold: the seeming arbitrariness of the sociological factors in software development is at odds with the formal and familiar technical aspects; and the lack of suitable tools with which to model and understand human dynamics.

However, these impediments may be overcome. For example, system dynamics is a modelling approach to dynamic socio-technical problems, stemming from the work of Forrester[20,21,22] at MIT and since developed[36,39,43], that allows a modeller to mix soft variables (morale, perceptions, motivations) with familiar hard variables (time, cost, resources). A system dynamics model is not so much a tool for time-point prediction, but more of an experimental device to see how certain variables might change over time under the influence of unappreciated causal relationships, dynamic complexity, and structural delays. The end result is hopefully a more informed mind set with which to manage the situation at hand[13].

By way of illustration, this paper presents some initial results of a system dynamics model based on Frederick Brooks'[11] well-known informal law which warns against adding more software developers to a late project for risk of making matters worse. Brooks' law, the crystallisation of many years of practical software project experience, has been critiqued many times in the literature and generally enjoys wide support, making it a solid basis for any model of the socio-technical aspects of software project management. However, it operates at a high level of aggregation and is most often associated with large-scale software development projects. In contrast, the system dynamics model presented here creates a small-team, small-project environment more likely to be encountered by software engineers in the current market.

Brooks Law

Frederick Brooks was an IBM programmer and hardware architect who in 1964 became the manager of IBM's OS/360 development. Then and now, OS/360 was one the largest and most complex operating systems ever attempted[6,27], and was a significant business risk for IBM given that it would not be backward-compatible with IBM's older machines[19,38]. Brooks' experiences on the OS/360 project and his observations of the industry in general are collected in his book *The Mythical Man-Month*[11,12]. The title refers to that fundamental unit of measurement and scheduling, the man-month; a unit that Brooks believes is often misunderstood:

> Cost does indeed vary as the product of the number of men and the number of months. Progress does not. Hence the man-month as a unit for measuring the size of a job is a dangerous and deceptive myth. It implies that men and months are interchangeable.[12]

Because of this lack of interchangeability, Brooks' informal law states that adding more developers to a late software project in the hope of meeting a looming deadline will only make matters worse. The reason lies in the fact that software projects often cannot be broken into isolated, independent units of work, meaning that the developers need to coordinate their activities at a detailed level. Therein lies an unappreciated communications overhead. For example, if a group of n developers need to coordinate their efforts with each other then the number of communication paths can be represented by $n (n - 1)/2$. Time spent navigating these paths is time not spent being directly productive.

When new developers are added to the equation, the communications overhead is amplified. The new developers are usually not immediately productive because they need to become acquainted with the overall aims of the project, its strategy and the general plan of work[10,37], and they possibly need to undergo some form of organisational socialisation[34]. The best, and often only, people able to provide this training and socialisation are the existing developers, who are in the process diverted from their primary tasks.

The net result is that more time is lost in bringing the new developers up to speed and in additional coordination efforts than is gained in productive time.

Brooks' law has an intuitive appeal and has been generally supported in the literature[7,15,17,41,45]. Writing recently, Brooks acknowledged that his law was a gross generalisation and yet, in the absence of anything more conclusive, it remained 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"[12].

However, not all would agree with this assessment. For example, the effects of Brooks' law can be actively mitigated by strategies such as adding developers early in the development cycle[3,26], adding more developers than are expected to be needed[24], and ensuring that documentation, technical reviews, and a less territorial ownership of software artefacts by individual developers are used to spread the knowledge about the project[28,42]. Raymond[30] even suggests that Brooks' law breaks down completely under large-scale, distributed development such as Linux.

So, what are students and practitioners to make of these different views? In many respects Brooks' law has stood the test of time but has perhaps been learned too well, becoming a mantra rather than a considered decision-making tool applicable to modern software development[28]. This will continue to be the case until it is turned into something more concrete than a rule-of-thumb, and some of its underlying assumptions are challenged. For example, most debate around Brooks' law accepts that the communications structure of software projects is a complete graph in which all developers need to talk each other, yet this need not be so. Creating a system dynamics model is one way of turning a rule-of-thumb into something more tnagible.

System Dynamics Model Description

The model described here has been built using a system dynamics software package called iThink (High Performance Systems, http://www.hps-inc.com/), the components of which are described more fully in the Appendix. The model describes a hypothetical software development project and makes a range of assumptions that will naturally vary according to local conditions. What is important is not so much the magnitude of these assumptions in this particular instance, but that they can be tuned to the environment they are modelling as needed.

Figure 1 shows the Human Resources section of the model which describes the hiring, assimilation, and resignation of software developers on the project. As new developers are recruited they enter the 'plumbing' of the model from the left and progress from being New Hires to Midrangers, and finally to Old Hands, reflecting their growing ability as they come up to speed with the project. The average time that a New Hire will take to progress to a Midranger and then an Old Hand has been set at two and four months respectively, meaning a new developer is expected to be fully productive after a total of six months[24].

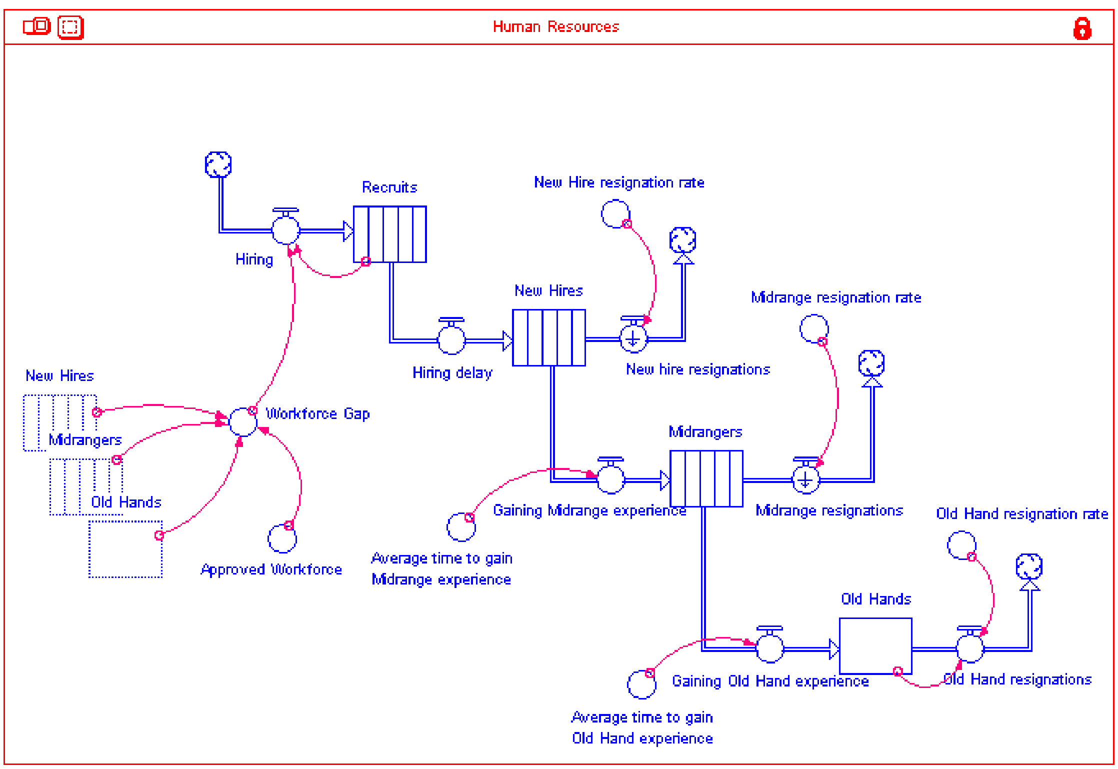


Figure 1. Human resources section of the model.

As might be expected, the project has an approved workforce level which reflects the amount of work to be done within the required time. Should the total number of developers fall below this approved level through resignations, then the process of hiring new staff is begun. However, this takes time and a delay of up to two months is not unreasonable between a position becoming available and it being eventually filled[1,37,40]. For simplicity, it is assumed that no New Hires will resign and the average resignation rate of Midrangers and Old Hands will be 5%[4,5].

Figure 2 shows the Productiveness section of the model.

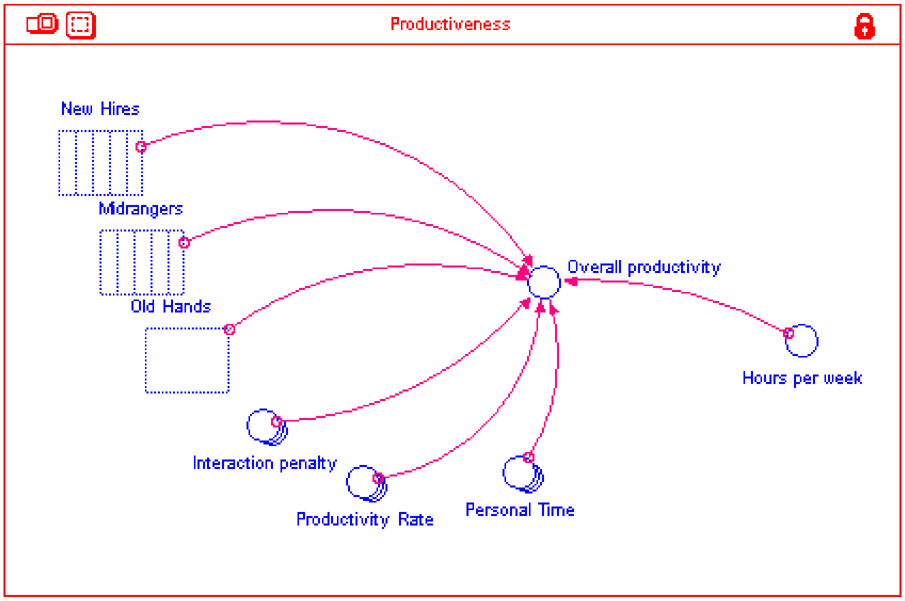


Figure 2. Productivity section of the model.

For the purposes here, productivity is considered to be potential productivity in the hours allowed during the working week, minus any losses due to faulty processes[2]. A faulty process might be excessive administrative duties, red tape, or demands for prolonged over-time, amongst other local factors. The model here considers only three basic factors: the interaction penalty discussed by Brooks, the varying levels of productivity between the New Hires, Midrangers, and the Old Hands; and an allowance that some of each developer's day may be occupied in personal pursuits. The assumptions behind these factors are summarised in Table 1.

The project to be modelled is made up of 8 developers of varying skills levels. New Hires are considered to be working at only 50% of their capacity during the time in which it takes them to come up to speed with the project, Midrangers are working at 75% capacity, while Old Hands are considered to be as productive as possible at 95%[24]. In addition each developer has an activity profile: net productive time during a working week is taken to be 100% of that possible, less unproductive personal time, set at a standard 10% of the working week[35], less the interaction penalty. The symmetric matrix to the side of table 1 represents the time in hours per week that developers spend coordinating their activities with other developers. In contrast with a key assumption behind Brooks' law, not all developers necessarily need to communicate with all other developers.

For example, Developer 1 is net productive for 77.5% of the working week, losing 10% of the week in personal time, 12.5% of the week coordinating activities with Developers 3, 4, 5, 6, and 7, and of that time is working at 50% effective capacity.

		Activity Profile (% of the working week)			Developer 1	Developer 2	Developer 3	Developer 4	Developer 5	Developer 6	Developer 7	Developer 8
	Productive Capacity	Net Productive	Personal Time	Interaction Penalty								
Developer 1	New Hire 50%	77.5	10.0	12.5	0	0	1	1	1	1	1	0
Developer 2	Midranger 75%	77.5	10.0	12.5	0	0	1	1	1	1	0	1
Developer 3	Midranger 75%	82.5	10.0	7.5	1	1	0	0	0	0	0	1
Developer 4	Midranger 75%	80.0	10.0	10.0	1	0	1	0	0	1	0	1
Developer 5	Old Hand 95%	77.5	10.0	12.5	1	1	0	0	0	1	1	1
Developer 6	Old Hand 95%	77.5	10.0	12.5	1	1	0	1	1	0	0	1
Developer 7	Midranger 75%	82.5	10.0	7.5	1	0	0	0	1	0	0	1
Developer 8	New Hire 50%	75.0	10.0	15.0	0	1	1	1	1	1	1	0

Table 1. Individual developer productive capacity and activity profiles.

The actual work to which the developers' productivity is applied is represented by the Development Work section of the model shown in Figure 3.

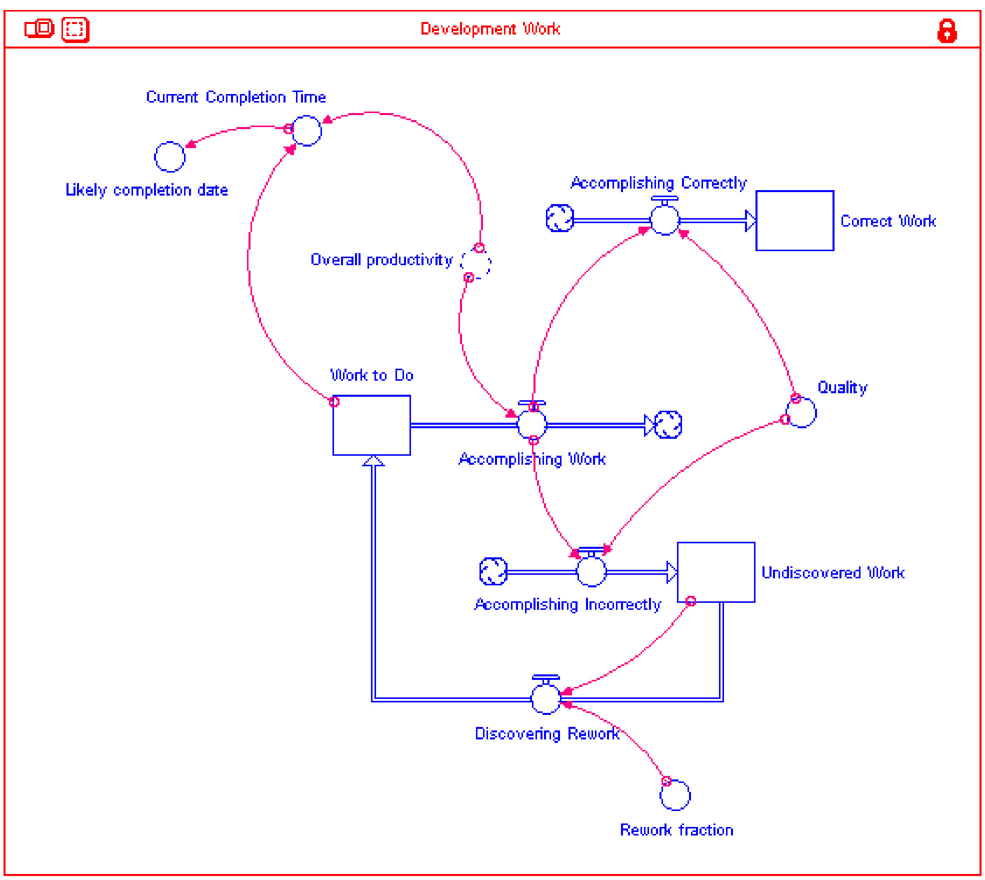


Figure 3. Development Work section of the model.

This section of the model broadly follows the classic project structure defined by Roberts[32,33]. The project starts with a certain amount of Work to Do measured in person-months. The overall productivity of the developers is applied to reduce this work, but in the process new work may be discovered because requirements have changed or the original specifications were incomplete, and some work already done may need to be reworked because mistakes have been made. Undiscovered work and the need for rework are influenced by many factors such as schedule pressure, the presence or absence of quality control and change control mechanisms, and general management of the project. In this simple model, these extraneous factors have not been modelled, and it is considered that 10% of all completed work will need to be reworked in some way. A Likely Completion Date is calculated by dividing the Work to Do by the Overall Productivity of the developers and adding it to the time already elapsed.

The project is complete when there is no more Work to Do or Undiscovered Work.

Running the Model

The hypothetical project modelled here has been sized at 90-person months which, using accepted cost-estimation tools such as COCOMO II[8], would take the eight developers about 12 months to complete.

Figure 4 shows the human resource numbers over a period of 24 months. The number of Old Hands gradually rises and the number of Midrangers gradually drops as the Midrangers gain experience. Likewise, the number of New Hires drops as they transition to become Midrangers.

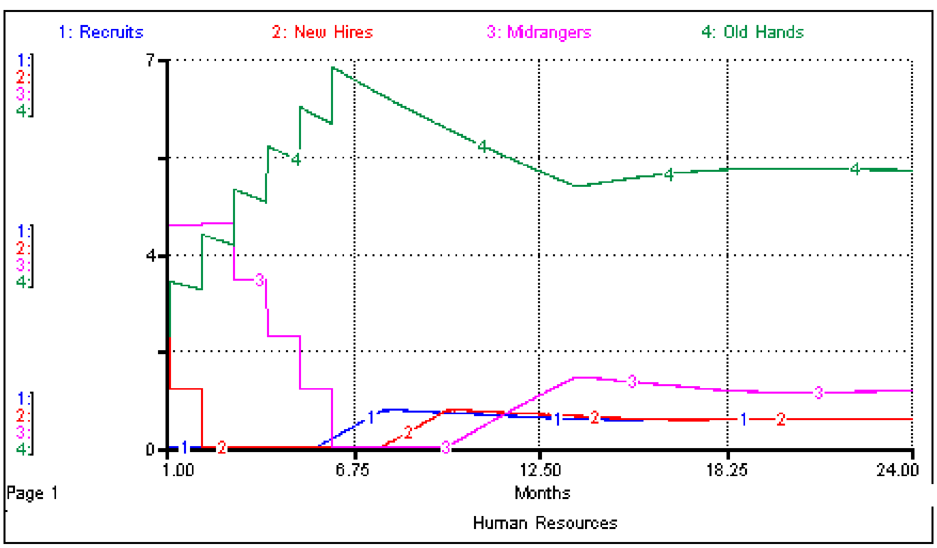


Figure 4. Human resource profile under the model's initial conditions.

Around the fourth month of the project, normal attrition (resignation of Midrangers and Oldhands) has meant that the total number of developers has fallen below the Approved Workforce of eight, and the hiring process is initiated. But, because of the hiring delay, the new developers don't make an appearance until around the seventh month.

Figure 4 also shows that the project settles down to a certain human resource profile: mainly Old Hands with a smaller number of Midrangers and New Hires, and a certain constant level of recruitment.

Under this human resource profile, the development proceeds as shown in Figure 5.

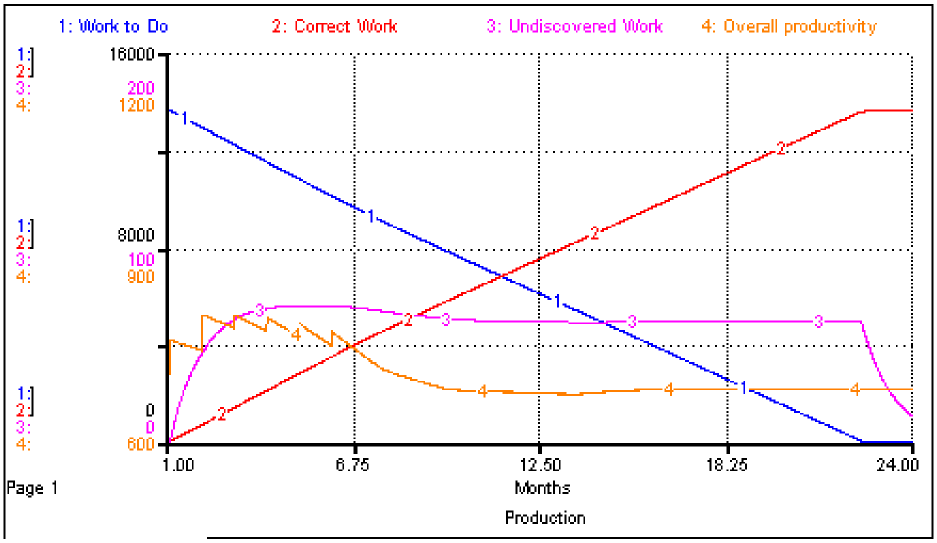


Figure 5. Development progress under the model's initial conditions.

Disturbingly, Figure 5 shows that the project will not be completed (no more Work to Do or Undiscovered Work) until just after the twenty-fourth month, double the original estimate.

To test Brooks' law, it is surmised that in the eighth month the development manager realises the project will not be completed within its scheduled period of 12 months and therefore decides to hire an additional four developers. The project under these circumstances is shown in Figures 6 and 7.

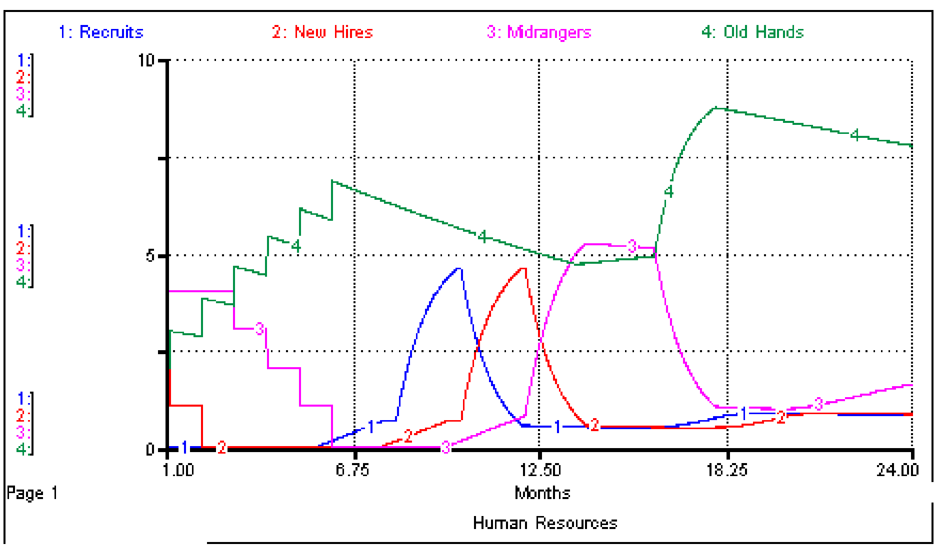


Figure 6. The human resource profile showing the hiring of four additional developers after the eighth month of the project.

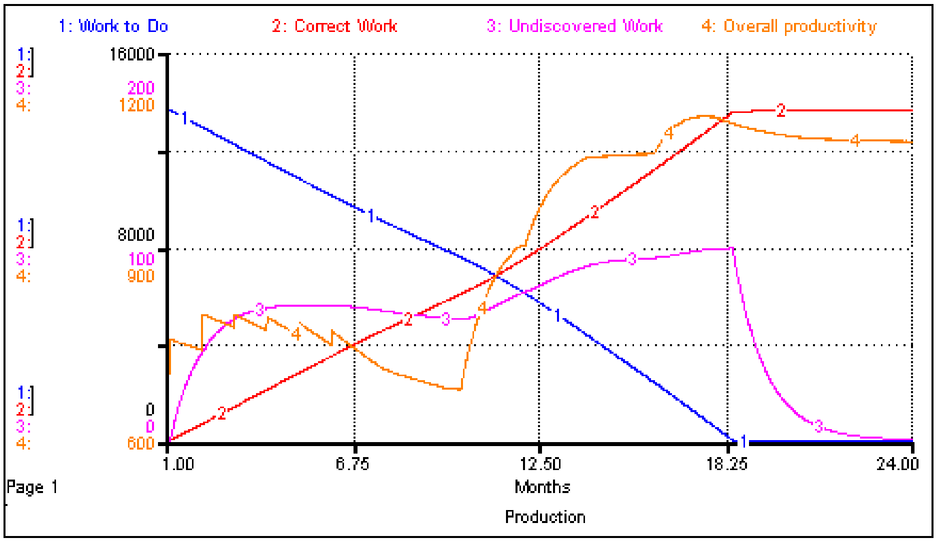


Figure 7. Development progress with four new developers joining after the eighth month.

Productivity begins to rise in the tenth month as the New Hires join the project, yet the overall effect of increasing the development workforce by 50% has been to bring the project's completion date forward only marginally. In fact, setting out to double the workforce after the eighth month has the effect of only bringing the completion date forward only one more month to that shown in Figure 7.

Given the parameters of this model, Brooks' law is not fully supported. Under a human resource profile that acknowledges that not all developers contribute equally to the project all the time, and which does not assume complete communications between all developers, perhaps Brooks' law could be rephrased as:

> *Adding more developers to a late project may not make the project later, but doing so will be of only marginal assistance.*

Indeed, if the project had been realistically sized and resourced at the start, then the need to consider changes mid-stream may not be needed.

System Dynamics Model Validation

George Box has famously said that all models are wrong, but some are useful. The reason that models are wrong is that they are necessarily selective abstractions of reality: just as a map as detailed as the landscape it described would be as big as the landscape itself (and of no use), a model that perfectly replicated a system under study would serve no purpose[9]. Even so, models can be useful:

> Models have this merit, that they do not allow us to comfort ourselves with the notion that we are following up an "idea" when we are only moving from one observation to the next in the hope that something will turn up. Too often the hypotheses with which we work are at home in the twilight regions of the mind, where their wavering outlines blend into a shadowy background. There they are safe from sudden exposure, and are free to swoop down for sustenance on whatever datum comes their way. Models are at any rate conscious, explicit, and

> definite; there is nothing ghostly in their appearance or manner; they look healthy even up to the very moment of their death… The model saves us from a certain self-deception. Forced into the open, our ideas may flutter helplessly; but at least we can see what bloodless creatures they are. As inquiry proceeds, theories must be brought out into the open sooner or later; the model simply makes it sooner.[25]

When a model becomes more than a mental model, it has a form that allows it to be shared, discussed, and hopefully improved upon; yet, it must be able to demonstrate a degree of validity for this to happen.

The compass of a system dynamics model, such as the one of Brooks' law discussed here, means that the rules by which it is validated will be slightly different from other modelling techniques. For example, the output of a system dynamics model is meant to be read, not for particular time-point predictions, but for qualitative behavioural patterns such as growth, decline, oscillation, stability, and instability[29]. This goal of understanding general dynamic tendencies means that the model's parameters are less reliant on highly precise numerical data. Furthermore, the long-term nature of system dynamic problem statements means that parameters are likely to exceed historic ranges in any case; while the non-linear feedback structure of the models makes them less sensitive to precise parameter changes.

System dynamics models also make room for soft variables such as degrees of motivation, perception, understanding. For those familiar with models based on more demonstrable data certainty, including these soft variables may seem to threaten the integrity of the model. Yet:

> As long as the purpose of your model is not to predict the numerical magnitude of particular soft variables, you can greatly benefit from including them in your models. Doing so will cause you to think in a rigorous manner about the relationships the variables bear to other variables in the system.[31]

The calibration of soft variables may also seem an arbitrary process in which the model is 'made' to respond in a certain manner. However, the way in which the soft (and hard) variables react must be internally consistent, that is, they must generate behaviour that matches what is observed in the actual system[23,29,31].

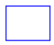
Conclusions

The results of this model are at variance with Brooks' law, but this might be expected because the model attempts to more realistically reflect the profile of current software development projects. For example, not all developers should be considered to be equally or immediately productive, and it need not be the case of each developer needs to coordinate their activities with each other developer. Nevertheless, the effect of adding more developers to the project only seems to help in a marginal way suggesting that there is some constraining force at work. If a software development project seems unlikely to meet its published completion date, then the common practice of adding more resources may not be the solution. Rather than attempting mid-course corrections, correctly sizing and resourcing projects from the start would appear to be a more appropriate solution and is the subject of continuing research.

While just one interpretation of the human dynamics of software project management, the system dynamics model discussed here is a means of further exploring the domain and hopefully contributing to the more rounded professional development of software engineers.

Appendix: The Language of iThink

System dynamics models described by iThink use the following grammatical elements:

- Stocks,  , are the nouns of iThink. They represent an accumulation of something at a particular point in time. The slatted stocks used in the model of Brooks' law are a special version known as conveyors. They work in the same way as regular stocks except that anything entering the conveyor rides along it for a set period of time and then leaves.

- Flows, , are the verbs of iThink. Stuff (information, material, staff, money…) flows through the pipe of the flow in the direction of the arrow and at a rate determined by the flow regulator in the middle. The flow regulator is fitted with a spigot that can be conceptually tightened or loosened by other variables within the model. The cloud at the end of the flow represents the boundary of the model.

- Converters,  , can be thought of as adverbs that modify flows. They are often used to break out the detail of logic, that might otherwise be buried within a flow, and might be used to represent constant values. These typically influence the behaviour of the regulators on the flows.

- Connectors, , tie the other three building blocks together. They represent inputs and outputs, not inflows and outflows. Connectors do not take on numerical values: they merely transmit values taken on by other building blocks.

Because iThink models can quickly become cluttered, any model element can be 'ghosted'. For example, in the Productiveness section of the Brooks' law model, the stocks New Hires, Midrangers, and Old Hands have been 'ghosted' (indicated by dotted outlines) rather than drawing connectors from the Human Resources section. The aim is to keep the model depiction clear and simple.

References

1. Abdel-Hamid, T. K. (1989). 'The Dynamics of Software Project Staffing: A System Dynamics Based Simulation Approach.' *IEEE Transactions on Software Engineering*, vol. 15, no. 2 (February), p. 308 – 318.

2. Abdel-Hamid, T. K. (1989). 'A Study of Staff Turnover, Acquisition, and Assimilation and Their Impact on Software Development Cost and Schedule.' *Journal of Management Information Systems*, vol. 6, no. 1 (Summer), p. 21 – 40.

3. Abdel-Hamid, T. K. and Madnick, S. E. (1991). *Software Project Dynamics: An Integrated Approach*. Englewood Cliffs: Prentice-Hall.

4. Bartol, K. M. (1983). ‘Turnover Among DP Personnel: A Casual Analysis.’ *Communications of the ACM*, vol. 26, no. 10 (October), p. 807 – 811.

5. Bartol, K. M. and Martin, D. C. (1982). ‘Managing Information Systems Personnel: A Review of the Literature and Managerial Implications.’ *MIS Quarterly*, vol. 6, no. 5 (December), p. 49 – 70.

6. Belady, L. A. and Lehman, M. M. (1976). ‘A Model of Large Program Development.’ *IBM Systems Journal*, vol. 15, no. 3, p. 225 – 252.

7. Boehm, B. W. (1981). *Software Engineering Economics*. Sydney: Prentice-Hall.

8. Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R. J., Reifer, D. J. and Steece, B. (2000). *Software Cost Estimation with Cocomo II*. Upper Saddle River: Prentice Hall.

9. Bonini, C. P. (1963). *Simulation of Information and Decision Systems in the Firm*. Englewood Cliffs: Prentice-Hall.

10. Bradley, J. and McGrath, G. M. (2000). ‘Boot Camp or Bordello: Whipping Rookies into Shape.’ *Proceedings of the Twenty First International Conference on Information Systems*, (Brisbane), p. 467 – 472. Association for Information Systems, Atlanta, GA, USA.

11. Brooks, F. P. (1982). *The Mythical Man-Month: Essays on Software Engineering*. Reading: Addison-Wesley.

12. Brooks, F. P. (1995). *The Mythical Man-Month: Essays on Software Engineering,* 20th anniversary edition. Sydney: Addison-Wesley.

13. Caulfield, C. W. and Maj, S. P. (2002). ‘A Case for System Dynamics.’ *Global Journal of Engineering Education*, vol. 6, no. 1, p. 25 – 34.

14. Constantine, L. L. (1995). *Constantine on Peopleware*. Englewood Cliffs: Yourdon Press.

15. Davis, A. M. (1995). *201 Principles of Software Development*. Sydney: McGraw-Hill.

16. DeMarco, T. (1991). ‘Non-Technological Issues in Software Engineering.’ *Proceedings of the 13th International Conference on Software Engineering*, (Austin, Texas), p. 149 – 150. Los Alamitos: IEEE Computer Society Press.

17. DeMarco, T. (1997). *The Deadline: A Novel About Project Management*. New York: Dorset House Publishing.

18. DeMarco, T. and Lister, T. (1999). *Peopleware: Productive Projects and Teams,* 2nd edition. New York: Dorset House Publishing Co.

19. Evans, B. O. (1986). ‘System/360: A Retrospective View.’ *IEEE Annals of the History of Computing*, vol. 8, no. 2 (April – June), p. 155 – 179.

20. Forrester, J. W. (1961). *Industrial Dynamics*. Waltham: Pegasus Communications.

21. Forrester, J. W. (1969). *Urban Dynamics*. Portland: Productivity Press.

22. Forrester, J. W. (1971). *World Dynamics*. Portland: Productivity Press.

23. Forrester, J. W. and Senge, P. M. (1980). ‘Tests for Building Confidence in System Dynamics Models.’ In A. A. Legasto, J. W. Forrester and J. M. Lyneis (eds.), *System Dynamics*, (1980), pp. 209 - 228. New York: North Holland.

24. Gordon, R. L. and Lamb, J. C. (1977). ‘A Close Look at Brooks' Law.’ *Datamation*, vol. 23, no. 6 (June), p. 81 – 86.

25. Kaplan, A. (1973). *The Conduct of Inquiry: Methodology for Behavioral Science*. Aylesbury: Intertext Books.

26. McCarthy, J. (1995). *Dynamics of Software Development*. Redmond: Microsoft Press.

27. McConnell, S. (1999). *After the Gold Rush*. Redmond: Microsoft Press.

28. McConnell, S. (1999). 'Brook's Law Repealed.' *IEEE Software*, vol. 16, no. 6 (November/December), p. 6 – 8.

29. Meadows, D. H. and Robinson, J. M. (1985). *The Electronic Oracle: Computer Models and Social Decisions*. New York: John Wiley & Sons.

30. Raymond, E. S. (2001). *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Sebastopol, California: O'Reilly & Associates.

31. Richmond, B. (1999). 'Modelling "Soft" Variables.' *An Introduction to Systems Thinking*, pp. 9-1 - 9-10. Hanover: High Performance Systems.

32. Roberts, E. B. (1962). *The Dynamics of Research and Development*. Unpublished PhD dissertation, Massachusetts Institute of Technology, Cambridge, Massachusetts.

33. Roberts, E. B. (1981). 'A Simple Model of R&D Project Dynamics.' In E. B. Roberts (ed.) *Managerial Applications of System Dynamics*, pp. 293 – 314. Waltham: Pegasus Communications.

34. Schein, E. H. (1980). *Organizational Psychology,* 3rd edition. Englewood Cliffs: Prentice-Hall.

35. Scott, R. F. and Simmons, D. B. (1975). 'Predicting Programming Group Productivity— A Communications Model.' *IEEE Transactions on Software Engineering*, vol. 1, no. 4 (December), p. 411 – 414.

36. Senge, P. M. (1990). *The Fifth Discipline: The Art & Practice of The Learning Organization*. Milsons Point: Random House.

37. Sengupta, K., Abdel-Hamid, T. K. and Bosley, M. (1999). 'Coping with Staffing Delays in Software Project Management: An Experimental Investigation.' *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 29, no. 1, p. 77 – 91.

38. Stallings, W. (1992). *Operating Systems*. New York: Macmillian Publishing Company.

39. Sterman, J. D. (2000). *Business Dynamics: Systems Thinking and Modelling for a Complex World*. New York: Irwin McGraw-Hill.

40. Stone, R. J. (1998). *Human Resource Management,* 3rd. Brisbane: John Wiley & Sons.

41. Weinberg, G. M. (1992). *Quality Software Management: Volume 1 Systems Thinking*. New York: Dorset House Publishing.

42. Weinberg, G. M. (1998). *The Psychology of Computer Programming,* silver anniversary edition. New York: Dorset Housing Publishing.

43. Wolstenholme, E. F. (1990). *System Enquiry: A System Dynamics Approach*. Brisbane: John Wiley & Sons.

44. Yourdon, E. (1992). *Decline and Fall of the American Programmer*. Sydney: Prentice-Hall.

45. Yourdon, E. (1997). *Death March: Managing "Mission Impossible" Projects*. Upper Saddle River: Prentice Hall.

46. Yourdon, E. (1998). *Rise and Resurrection of the American Programmer*. Sydney: Prentice-Hall.

Biographies

CRAIG CAULFIELD graduated from Murdoch University in Perth, Australia in 1994 with a Bachelor of Science in computer science and completed a Masters of Science in software engineering in 2001 through Edith Cowan University in Perth, Australia. He currently works as a senior software developer for a large Australian agribusiness while studying towards a PhD in computer science at Edith Cowan University.

GURPREET KOHLI is a PhD student at Edith Cowan University with two years of experience in Lecturing and Developing Network and Data Communication units at Edith Cowan University. Gurpreet is currently looking into web services and capacity planning of e-business sites as part of his research at Edith Cowan University.

S. PAUL MAJ is a senior academic at the School of Computer and Information Science, Edith Cowan University, Perth, Australia, and also Adjunct Professor at the Department of Information Systems and Operations Management, University of North Carolina (Greensboro) in the USA. He is an internationally recognised authority in laboratory automation and has published a commissioned book in this field.