*Global J. of Engng. Educ., Vol.6, No.1*
*Published in Australia*

*© 2002 UICEE*

# A Case for System Dynamics*

**Craig W. Caulfield**
**S. Paul Maj**
*Edith Cowan University, 2 Bradford Street, Mount Lawley, Perth, WA 6050, Australia*

Engineering education provides a thorough and systematic training in the design, development, maintenance and management of complex technical systems. While such education provides the necessary technical depth to graduates, many technical systems are best understood from the perspective of human and socio-economic relationships. A case in point may be Fred Brooks' law that states adding more developers to a late software engineering project will only make it even more behind schedule. Brooks' law is based on the understanding that additional, new software engineering staff will need time to come up to speed with the project and in doing so will divert the existing developers from their primary tasks. While Brooks' law is intuitively appealing, students and practicing software engineers really have no way of testing its efficacy in their particular situations. A tool to overcome this difficulty may be system dynamics. System dynamics is a systems thinking methodology for building quantitative and qualitative models of complex situations so that they can ultimately be better understood and managed. Accordingly, it can be argued, that system dynamics should be an essential part of the education of engineers from most, if not all, of the major disciplines.

## INTRODUCTION

Engineering education can deliver training that is all-inclusive and systematic in the design, development, maintenance and management of intricate technical systems. Without question, such education provides the necessary technical depth to graduates. However, many technical systems are best understood from the perspective of human perceptions and also that of a wider socio-economic context. It has been well documented that the success of technical projects is quite often almost entirely dependent on these factors.

It is a curious paradox that the software industry has helped provide the means by which others have been able to automate, reengineer and economy-scale their businesses, that is, reduce the human variable, and yet remains itself very people sensitive and intensive. For example:

*Highly skilled people with appropriate experience, talent, and training are key to producing software that satisfies user needs on time and within budget. The right people with insufficient tools, languages, and process will succeed. The wrong people (or the right people with insufficient training or experience) with appropriate tools, languages, and process will probably fail* [1].

Tom DeMarco, co-author of the often-cited *Peopleware*, has found that most software development managers agree with this premise that a project's sociology will contribute more to the final outcome than the project's technology [2]. Sociology, in this context, means addressing issues such as team formation and dynamics, role assignment, hiring, motivation, workplace design, training and many other peopleware practices. However, the same managers do not conduct their projects with this regard and instead focus on that aspect they are most comfortable with: technology;

*The evident reason for this is that the manager knows how to do technology, but not how to do sociology. He/she doesn't know how to manage* [3].

*A revised and expanded version of a keynote address presented at the 3rd *Asia-Pacific Forum on Engineering and Technology Education*, held in Changhua, Taiwan, from 8 to 11 July 2001. This paper was awarded the UICEE diamond award (first grade) by popular vote of Forum participants for the most significant contribution to the field of engineering education.

One of the golden rules of software engineering texts maybe a case in point - Fred Brooks' informal law that states that adding more software developers to a late project will only make it later [4]. Brooks' law is based on the understanding that the new developers will need time to come up to speed with the project and in doing so will divert the existing developers from their primary and now critical tasks. While Brooks' law is intuitively appealing, students and practicing software engineers really have no way of testing its efficacy in their particular situations because such systems are difficult to model.

One possible way to address such situations is by using the systems thinking methodology, system dynamics.

System dynamics is concerned with building quantitative and qualitative models of complex problem situations and then experimenting with and studying the behaviour of these models over time. Often such models will demonstrate how unappreciated causal relationships, dynamic complexity and structural delays may lead to counter-intuitive outcomes of less-informed efforts to improve the situation. System dynamics models make room for soft factors such as motivation and perceptions so that engineering projects can ultimately be better understood and managed.

This paper presents some initial results of implementing a simple model of Brooks' law using a system dynamics modelling software package called *iThink* to support the argument that system dynamics should be an essential part of the education of engineers from most, if not all, of the major disciplines. The model is then extended beyond Brooks' exact scope to demonstrate how it might be possible to incorporate and validate soft variables alongside the more traditional variety.

## SYSTEM DYNAMICS

In the late 1950s, Jay Forrester of the Sloan School of Management at the Massachusetts Institute of Technology (MIT) was asked by General Electric to review the operations of their Kentucky appliance parts plant. The company was concerned about the oscillating nature of their production cycles that often saw periods of intense activity followed by times of virtual dormancy during which workers had to be laid off. Fluctuating demand and normal business cycles did not seem to adequately explain the situation. Coming from an electrical engineering background and with a keen interest in management science, Forrester approached the problem systematically, but with just a pencil and a note pad. Starting with columns for inventory, employees and orders, and factoring in:

> *...the policies they were following, one could decide how many people would be hired in the following week. This gave a new condition of employment, inventories, and production* [5].

Forrester's calculations amounted to a simulation of the system operating at General Electric's plant.

Stemming from this first analysis came an article for the *Harvard Business Review* in 1958 entitled *Industrial Dynamics - A Major Breakthrough for Decision Makers* with the theme being developed and expanded in the seminal work, *Industrial Dynamics* [6]. Industrial dynamics became system dynamics as it came to be used in areas other than industry.

For some time following the publication of *Industrial Dynamics*, system dynamics was used as a tool for looking at big-picture issues such as urban decay, major sociological conditions and world economics [7-9]. In more recent times, system dynamics has come back from the big end of town and has been finding a purpose for itself in a range of business and social applications. Instrumental in this change have been Peter Senge's *The Fifth Discipline* [10], and the development of intuitive, graphical software packages that have made system dynamics modelling more democratic by hiding the computer source-code look of traditional models. As a measure of this democracy, system dynamics now finds a place for itself in a number of primary and secondary schools in the United States of America, Australia and Europe, well beyond its ground zero at MIT.

To more formally define system dynamics, it could be said that it:

> *...is concerned with creating models or representations of real world systems of all kinds and studying their dynamics (or behaviour). In particular, it is concerned with improving (controlling) problematic system behaviour... The purpose in applying System Dynamics is to facilitate understanding of the relationship between the behaviour of the system over time and its underlying structure and strategies/policies/decision rules* [11].

A key element of this definition is the need to build a computer model of the system under consideration. The model is used to help understand the patterns of change or dynamics that a system exhibits over time and to identify the conditions that cause these patterns to be stable or unstable. This knowledge of the system can then suggest what kinds of prescriptions for governing it will work and what kinds may not [12].

However, building system dynamics models demands persistence. Translating real-world information into model elements is still an inexact science - trial and error can be just as valid as considered judgement based on experience. Perhaps a useful parallel can be drawn with that other hard, inexact activity: finding object-oriented classes. Bjarne Stroustrup, the creator of C++, notes that in design and programming there are no cookbook methods that can replace intelligence, experience and good taste; *even he just tries things* [13]. The lesson for system dynamics modellers would seem to be the same: just start, try things, take advice of experienced modellers and then iterate, iterate, iterate.

Yet the effort of building a system dynamics model has some benefits including:

- Modelling brings about an understanding of the system because of the analytical and critical thinking process it calls for. It helps bring to the surface the mental models driving the current situation - those models

  > *...that one carries around in one's head for dealing with a problem or situation. Such a model maybe based on experience or intuition, or on folklore and myth; it may be influenced by politics and a wide spectrum of human emotions* [14].

  Mental models may also be totally inappropriate or counter-productive, or equally priceless. But unless they are turned into something more tangible, one may never know.
- System dynamics models make room for both quantitative or hard variables, being things that can be measured directly like program size, staffing numbers or dollars spent; and qualitative or soft variables such as motivation, commitment, confidence or perceptions. Soft variables have traditionally been left out of engineering models because they are difficult to measure and their importance may have been underestimated. Yet,

  > *...if you omit soft variables you run the risk of failing to capture something essential to driving human affairs. Leaving out something so essential is the only hypothesis that you can reject with absolute certainty!* [15].

  A system dynamics model can therefore be more informed about its problem space.

With a system dynamics model in hand and George Box's tongue-in-cheek caution in mind (all models are wrong, but some are useful), the model can be run. Certain variables can be held steady while others are changed, it can be placed under stress and tested for sensitivities and leverage points. In short, the model can be experimented with to better understand the present situation and to search for alternatives for improvement. It has been stated that:

> *The alternatives may come from intuitive insights generated during the* [initial analysis], *from experience of the analyst, from proposals advanced by people in the operating system* [or in the] *experience, art, and skill for imagining the most creative and powerful policy alternatives* [16].

Peter Senge points out that the causes of many problems

> *...lay in the very well-intentioned policies designed to alleviate them. These problems were actually* systems *that lured policy makers into interventions that focused on obvious symptoms not underlying causes, which produced short-term benefit but long term malaise, and fostered the need for still more symptomatic interventions* [10].

By simulating a problem space using a system dynamics model, it is possible to potentially make more informed decisions about events beyond our bounded rationality safe from the dangers of real-world experimentation.

## BROOKS' LAW

During the 1950s and early 1960s, Fred Brooks worked for IBM as a programmer and hardware architect. In 1964, he became the manager of IBM's Operating System/360 development, a large-scale and complex project intended to provide IBM's mainframe computers with a leading-edge operating system. To give an idea of the size of the project:

> *...the initial Windows NT project required about 1,500 staff-years of effort, but the development of IBM's OS/360, which was completed in 1966, required more than three times as much effort* [17].

His experiences, frustrations and joys during this time, and his observations of the wider industry after moving to the University of North Carolina, are embodied in the collection of essays *The Mythical Man-Month* [4]. 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* [4].

His law that states adding more software developers to an already late project will only make the problem worse is based on this lack of interchangeability of manpower and time. The cause lies in two areas:

- The new developers will need to be acquainted with the overall aims of the project, its strategy and the general plan of work. During this time, the new developers will not be full contributors and will likely divert the existing developers away from their primary tasks.
- If a group of developers, *n*, need to coordinate their efforts with each other then the number of communication paths can be represented by $n(n-1)/2$. This represents an interaction overhead, which may be realised in the form of project meetings, technical walkthroughs and complying with any progress reporting requirements.

Brooks' law is intuitively appealing and is generally supported in the literature [14][18-20]. Writing in the 20th anniversary edition of *The Mythical Man-Month* in 1995, Brooks acknowledged that his law was outrageously simplified, yet he still felt that it was 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* [4].

Yet, turning Brooks' law into something more than a rule of thumb, it should be able to be tested whether it is a useful concept outside the large-scale big business and government projects Brooks' was most familiar with.

## MODEL EXPLANATION

The following model of Brooks' law has been created using a system dynamics modelling package called *iThink*. The grammar of *iThink* consists of only four basic elements (stocks, flows, rates and connectors) and is largely intuitive so it will not be expanded upon here. Further details are provided in Appendix 1.

In addition, a range of assumptions is made 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 are relevant to the problem space under consideration and that they can be changed as needed.

Looking to the model, there is a hypothetical software development project in hand that has been estimated at 36-man months, or 6,240 hours, and must be completed within six months. To meet this deadline a staffing level of six developers has been approved. However, the project starts with only five developers, three of whom are experienced, meaning they are aware of the objectives of the project and the plan of work; and two who are new-hires. It is assumed that the new-hires will only be half as productive as their colleagues but will gradually come up to speed as they are assimilated. This transitioning from new-hires to experienced developers has been set at three months.

Recruiting is under way to bring the team up to full strength but advertising the position, assessing the applicants and making a decision all takes time. Therefore, a delay of some two months is not unreasonable [14]. At the same time, staff are likely to leave. For the purposes of this model, it is assumed that the average employment time will be nine months and, for simplicity, it is assumed that developers will not quit the team before becoming experienced developers. Figure 1 represents to model to this stage.

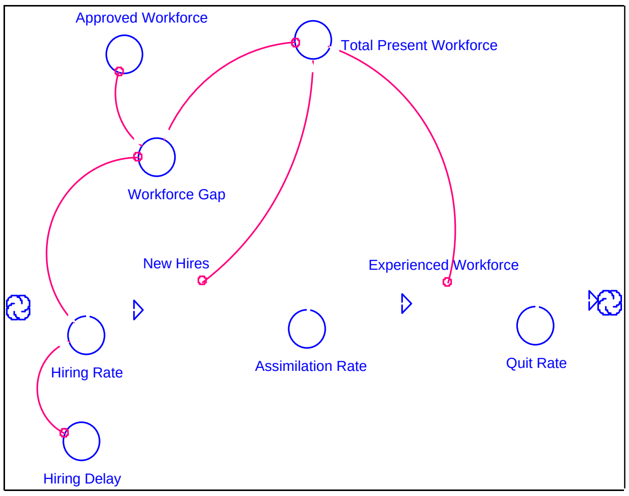


Figure 1: Model for personnel development in the project.

Staff enter the *plumbing* of the *iThink* diagram from the left, progressing to the right as they pass from being new-hires to experienced developers until they perhaps eventually leave the team. The *Total Present Workforce* will therefore be the sum of the two groups of developers. If the *Total Present Workforce* is less than the *Approved Workforce*, a *Workforce Gap* will exist and the hiring process will be initiated, subject to the prescribed delay of two months. Figure 2 represents the workflow of the project.

The team has 36 man-months of work to complete, therefore at the start of the simulation *Remaining Work* will represent this amount. Work units will flow towards *Work Completed* at a rate determined by

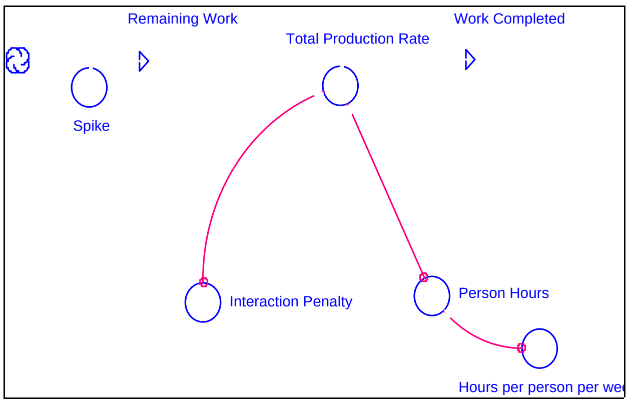


Figure 2: Workflow of the project.

the overall productivity of the team. Occasionally, there may be a spike in *Work Remaining* if the scope of the project is expanded or if the original work estimates have been found to be underestimated.

The total productivity of the team will be a function of the total workforce, the number of hours each person works per week, which has been set at a standard 40, the assumed productivity of the new-hires versus their more experienced colleagues and taking into account the interaction overhead required to coordinate all the individual development efforts. For the purposes of this model, it is assumed that the interaction overhead represents one hour per developer per week per communications path. If there are five developers, this equates to ten communications paths, and therefore ten hours per week per developer consumed in this over-head. The model in its entirety is represented by Figure 3.

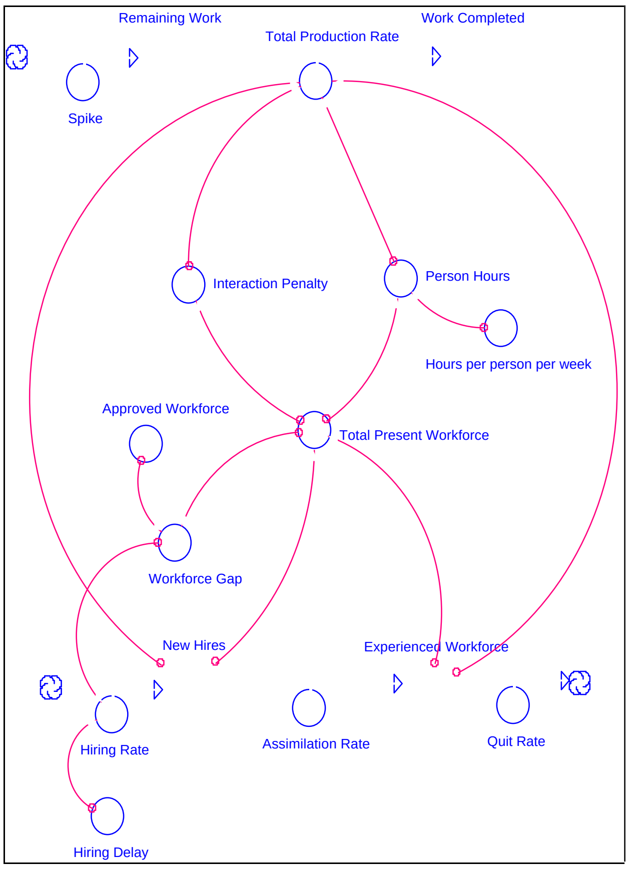


Figure 3: The model in its entirety.

## MODEL RESULTS

Setting the model to run under the initial conditions described above produces the graph in Figure 4.

The approved workforce consists of six developers, but at the start of the project only five are on hand.

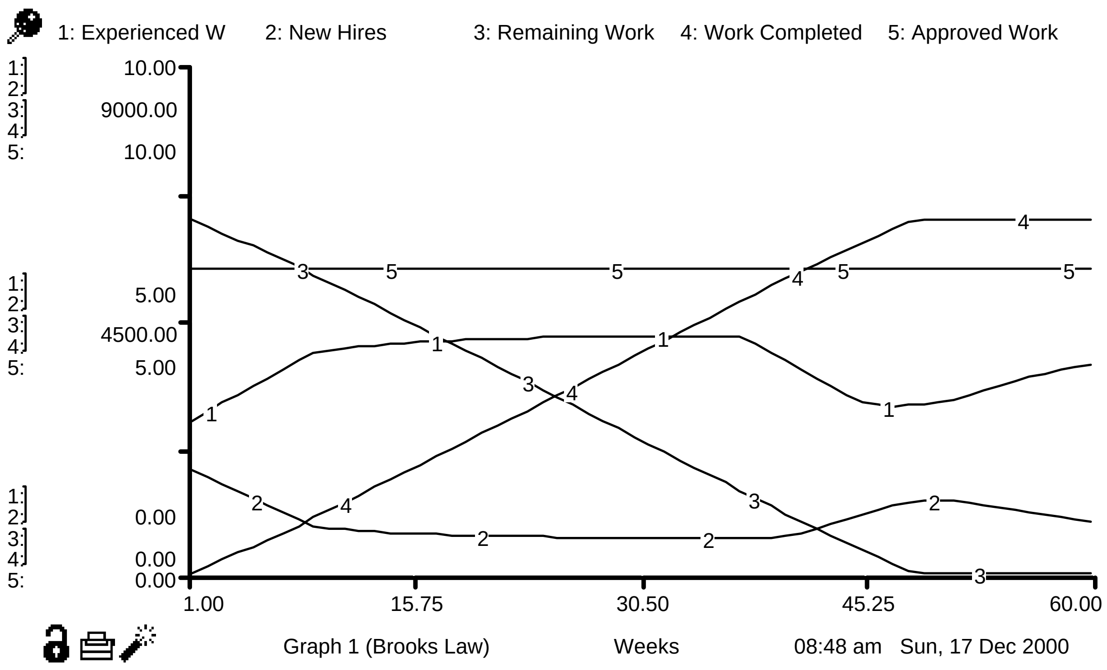


Figure 4: Graphical representation of the initial conditions.

The number of experienced workers gradually increases and the number of new hires dips as the latter come up to speed. The employment of one new developer, after the prescribed two-month delay, is masked in this transition. After nine months, or 36 weeks, experienced developers begin to leave, which initiates the hiring process again.

Even allowing for the fact that the project started with one developer less than required, the graph indicates that simply dividing the effort by the number of staff on hand will not yield an overall completion time. With the best will, the project will take nearly 12 months to complete rather than the original six.

Assume now that the project has been underway for five months, or 20 weeks, when it is discovered the original man-month estimates were understated.

Another 12 man-months of work have been assessed. Assuming 40-hour weeks and a present staff of six developers, this means the project will be extended by another eight weeks. To bring this figure down, the project manager decides to increase the approved staffing to eight developers. The resulting graph under these circumstances is shown in Figure 5.

Despite bringing on more staff, the project is still not able to hit its revised completion date and now takes nearly 18 months to complete.

## ENHANCEMENTS TO BROOKS' LAW

The variables that make up the model of Brooks' law thus far are informed by the quantitative, or hard, data typical to an engineering project. Yet, it may be as relevant to consider such a project from a socio-technical point of view, raising the need to evaluate qualitative, or soft, data. For example, soft factors may need to be considered, such as morale, commitment and knowledge levels, alongside hard factors such as headcounts, dollars spent and deliverablesThis is because such factors can have an impact in areas such as productivity and hence completion times and cost.

As mentioned previously, system dynamics makes room for these soft factors. To demonstrate how this might be possible the model of Brooks' law has been extended to incorporate a number of soft variables such as occupational stress and stakeholders' perceptions of quality of the deliverables.

The relationship between occupational stress and job performance has been well documented, discussed and modelled [21-25]. A certain level of stress is

> *...healthy and enables employees to feel a sense of achievement and to get satisfaction from the job. However, if the amount of stress exceeds the optimum and starts to place excessive demands on the employee, the result will be lower performance. At this point, the employee loses the ability to cope, finds difficulty in making decisions and demonstrates erratic behaviour* [25].

Meanwhile, the way in which clients perceive the quality of the service they receive can be considered a key performance indicator that has implications for remuneration packages and other penalties or rewards defined in service level agreements. These factors are,

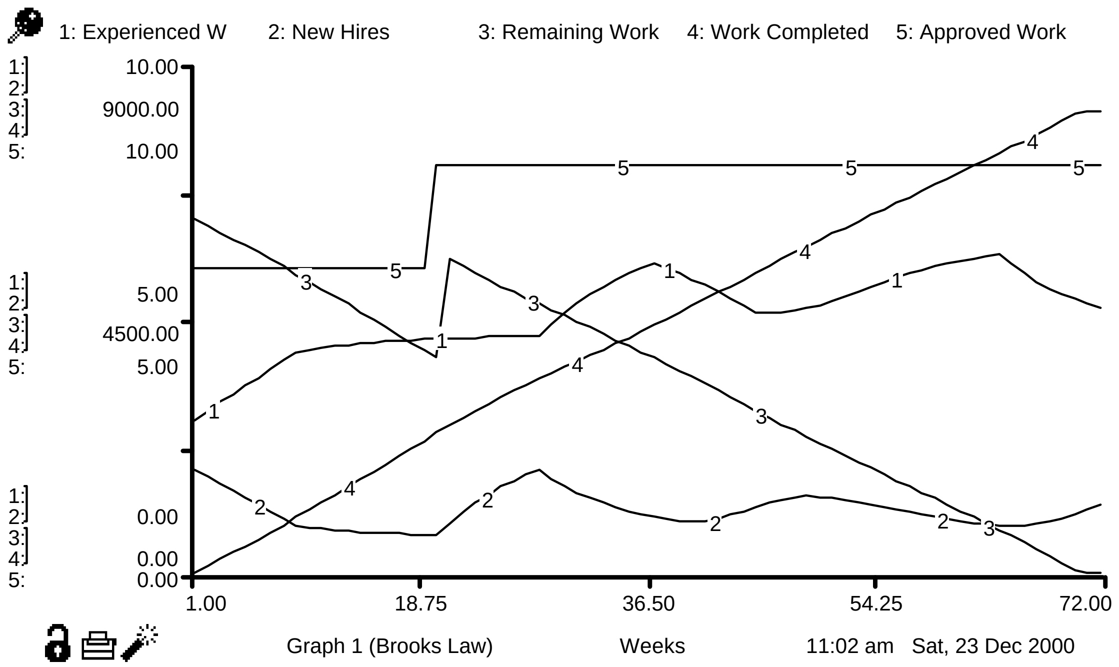


Figure 5: Graphical representation of the revised conditions.

therefore, considered as relevant additions to a model of a software development project.

The aim here is to show how it might be possible to incorporate qualitative factors, specifically perceptions of quality, into a model and then to show how it might be validated to the point where it could be used as a tool to influence policy decisions, despite lacking total quantitative comfort.

## PERCEIVED AND ACTUAL QUALITY

In many human transactions there is often a gap between perception and reality. Festinger discusses something similar in *A Theory of Cognitive Dissonance* [26]. Elements of an individual's cognition (things a person knows about themself, their behaviour, their surroundings) may deviate markedly from reality creating an uncomfortable dissonance. The cause of this dissonance may be imperfect knowledge about a situation or simply a factor of human society: *very few things are all black or all white; very few situations are clear-cut enough so that opinions or behaviours are not to some extent a mixture of contradictions* [26]. Furthermore, the dissonance may be fleeting or long-lasting; or an individual may be working to resolve and reduce the dissonance in some way or equally simply ignoring it. The result is that what we know and what we do may be inconsistent.

An example might illustrate the point. The quality of service experienced by a client, and therefore their perception of that quality, may be different for the actual quality being offered by the provider more generally. It could be the case that at a point in time, a client happened to encounter a staff member fully aware of their needs and were able to have their transaction completed quickly and efficiently. However, the rest of the provider's clients that day may not have been so lucky. Through incomplete knowledge a gap is created between perceived and actual quality.

The size of this gap can also grow, shrink and overshoot because there is often resistance, and therefore a delay, in adjusting perceptions and then taking action. A single experience of good or bad service may not cause a reaction, but an accumulation of such experiences will. The magnitude of this delay can be influenced by factors including the level of industry competition, client loyalty and mobility and the frequency of client contact [27].

Furthermore, this relationship will likely be asymmetric. When reality is less than perception, perceptions are adjusted rapidly as represented in Figure 6 (bad news travels quickly).

On the other hand, when reality is greater than perception, the adjustment time is much longer as represented in Figure 7 (it may take ten good experiences to overcome a single bad experience).

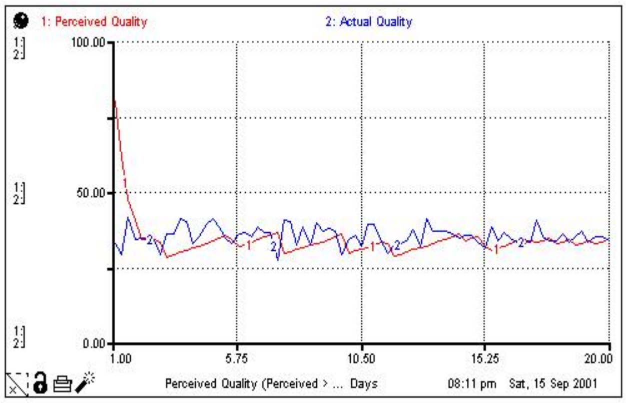


Figure 6: When actual quality is less than perceived quality, perceptions are quickly adjusted.

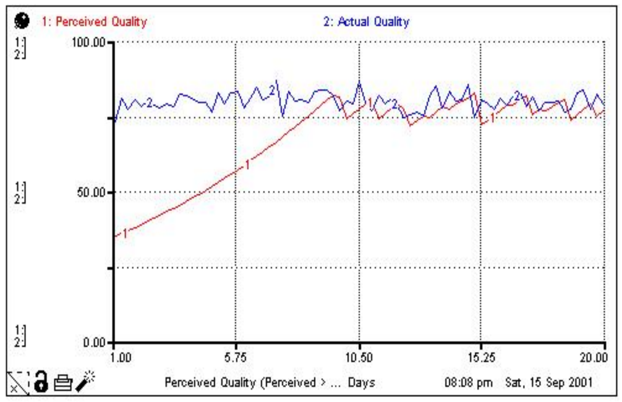


Figure 7: When actual quality is greater than perceived quality, perceptions are more slowly adjusted.

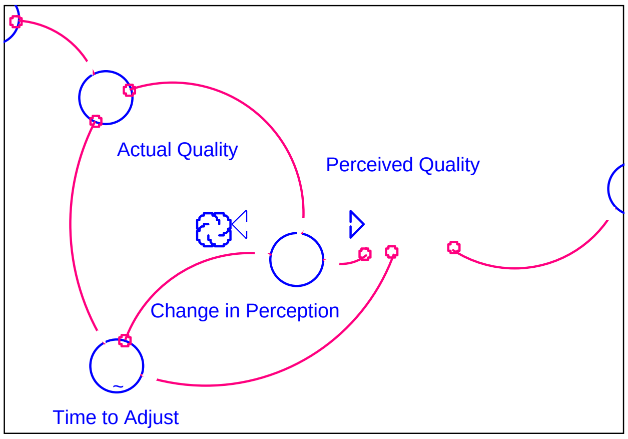


Figure 8: The service quality model component.

Within the model of Brooks' law, perceived and actual quality are exhibited in Figure 8.

The factors that determine quality have many interpretations [28]. However, for the purposes of this model, actual quality is taken to be a measure of the timeliness of the deliverables and the gap between the delivered functionality and the client's requirements

[29]. The dynamics of these variables influencing actual quality are outside the scope of this fragment of the model and are only shown as a generic inflow. Meanwhile, perceived quality is taken to be a subjective gauge of how the project's clients see the service level they are receiving. The inconsistency between actual and perceived quality and their relative levels will determine the rate at which the level of perceived quality will change in line with Figures 6 and 7.

Variables such as actual and perceived quality are soft factors that cannot be measured in the same way as physical quantities. So, for the purposes of modelling, these need to be quantified instead, that is, set them against an index of some kind [15]. In this case, 0 is taken to be a total absence of quality, while 100 is taken to be total fulfilment. At the start of the model, perceived quality is set to a value between 0 and 100 representing the current circumstances. As the dynamics of the model are played out over time, the levels of actual and perceived quality may rise and fall, in turn influencing other model variables.

For example, the level of quality perceived by the project's clients may influence future remuneration contracts and have broader market implications [29][30]. Internal to the project, this perception may influence the resources devoted to testing and quality procedures [31-33]. Again, such impacts appear outside the scope of this fragment of the model are shown as a simple outflow.

## VALIDATION

For those familiar with models based on more demonstrable data certainty, the treatment of soft variables such as occupational stress and perceptions of quality may seem to threaten the integrity of the final product. 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* [15].

Furthermore, the particular calibration of these relationships, and therefore the behaviour of the resulting model, will depend on the individual circumstances in which it is applied. For example, the present model assumes that instances of poor service will be quickly reflected in a declining perception of the quality of that service. In an industry with few repeat clients or long delays between client contacts, the delay in adjusting perceptions may be longer.

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 variables react must be internally consistent, that is, they must generate behaviour that matches what is observed in the actual system [15]. For example, if delivery deadlines are being consistently missed and required functionality is not being addressed, then the perceived level of service quality must decline. If the model produces behaviour contrary to this real-world pattern, then it needs to be reworked.

Sensitivity analyses designed to demonstrate internal consistency feature significantly amongst the range of tests that Forrester and Senge discuss through which a system dynamics model may be validated [34]. Importantly, these accepted tests focus more on *validating* rather than *proving* system dynamics models, on building confidence in a model's soundness and usefulness as a policy tool rather than rigorous time point predictions. The compass of a system dynamics model means that the rules by which it is validated will be slightly different.

Perhaps the ultimate test of any model is the quality of the decisions that result from it. It deserves mention that sometimes very few decisions flowed from some of the significant, early system dynamics modelling exercises [35-37]. These models tended to be large, complex and constructed by academics with only minimal involvement from the model's stakeholders beyond the initial problem definition and data collection.

Yet, as it is presently practiced, system dynamics is a very democratic and collaborative process. Sterman says that system dynamics is not a spectator sport by which he means involving the stakeholders early in the process and in doing so, giving them ownership of the model, is a critical success factor [38]. Furthermore, by making room for traditionally ignored soft variables and calibrating the variables according to real-world knowledge, by facilitating rather than creating in isolation, a more informed socio-technical model may be possible.

## CONCLUSIONS

The system dynamics model of Brooks' law presented here is necessarily generic and simplified and is part of ongoing research. But, even at this level, it is one realisation of a mental model that can now be shared, discussed, calibrated according to local circumstances and (hopefully) improved upon.

The results in this case tend to support Brooks' law that adding more software developers to an already late project will only make matters worse. However, this may not always be so. For example, using a more detailed model of Brooks' law, Abdel-Hamid and Madnick [31-33] found that if the developers are added early in the project rather than towards the end, the project will have more chance of hitting its deadlines. But, without the model, the belief that this might be so would have been without support.

Making system dynamics a part of all engineering disciplines would seem to be an incremental rather than a discontinuous step since engineers are likely to be already familiar with the benefits of building models. Typically these models have been informed by hard, quantitative data drawn from the model's domain. Also present in that domain may be softer, more qualitative, data that could be equally considered relevant to the model's outcome. System dynamics is one way of incorporating soft variables into models alongside the more traditional variety, while adding also its underlying theme that more informed socio-technical models are possible.

As a means of capturing mental models, building decision flight-simulators and communicating complex ideas at a higher level than verbal descriptions, system dynamics deserves serious consideration. In response, the methodology demands the patience to understand its concepts, nuances, and power.

## REFERENCES

1. Davis, A.M., *201 Principles of Software Development.* Sydney: McGraw-Hill (1995).
2. DeMarco, T. and Lister, T., *Peopleware: Productive Pro-jects and Teams* (2nd edn). New York: Dorset House (1999).
3. DeMarco, T., Non-technological issues in software engineering. *Proc. 13th Inter. Conf. on Software Engng.*, Austin, USA, 149-150 (1991).
4. Brooks, F.P., *The Mythical Man-Month: Essays on Software Engineering* (anniversary edn). Sydney: Addison-Wesley (1995).
5. Forrester, J.W., The beginnings of systems dynamics. Banquet talk given at the *Inter. Meeting of the System Dynamics Society*, Stuttgart, Germany, 13 July (1989), ftp://sysdyn.mit.edu/ftp/sdep/papers/D-4165-1.pdf
6. Forrester, J.W., *Industrial Dynamics.* Waltham: Pegasus Communications (1961).
7. Forrester, J.W., *Urban Dynamics*. Portland: Productivity Press (1969).
8. Forrester, J.W., *World Dynamics*. Portland: Productivity Press (1971).
9. Meadows, D.H., Meadows, D.L., Randers, J. and Behrens, W.W., *The Limits to Growth*. New York: Universe Books (1972).
10. Senge, P.M., *The Fifth Discipline: the Art and Practice of the Learning Organization.* Sydney: Random House (1990).
11. Wolstenholme, E.F., *System Enquiry: a System Dynamics Approach*. Brisbane: John Wiley & Sons (1990).
12. Stacey, R.D., *Strategic Management and Organisational Dynamics*. Melbourne: Pitman Publishing (1996).
13. Stroustrup B., *The C++ Programming Language* (2nd edn). Sydney: Addison-Wesley Publishing Company (1993).
14. Yourdon, E., *Rise and Resurrection of the American Programmer*. Upper Saddle River: Prentice Hall (1998).
15. Richmond, B., *Modelling "Soft" Variables*. An Introduction to Systems Thinking. Hanover: High Performance Systems, 9-1 - 9-10 (1999).
16. Forrester, J.W., System dynamics, systems thinking and soft OR. *System Dynamics Review*, 10, **2-3**, 245-256 (1994).
17. McConnell, S., *After the Gold Rush*. Redmond: Microsoft Press (1999).
18. Boehm, B., *Software Engineering Economics.* Upper Saddle River: Prentice Hall (1981).
19. DeMarco, T., *The Deadline: a Novel about Project Management.* New York: Dorset House (1997).
20. Pressman, R.G., *Software Engineering: a Practitioner's Approach* (4th edn). New York: McGraw-Hill (1997).
21. Selye, H., *Stress Without Distress*. Philadelphia: Signet Books (1974).
22. Homer, J.B., Worker burnout: a dynamic model with implications for prevention and control. *System Dynamics Review*, 1, **1**, 42-62 (1985).
23. Hooper, N., Coping with the modern 'madness'. *Business Review Weekly*, 17, **17**, 38-42 (1995).
24. Kramar, R., McGraw, P. and Schuler, R.S., *Human Resource Management in Australia*. South Melbourne: Addison Wesley Longman (1997).
25. Stone, R.J., *Human Resource Management*. Brisbane: John Wiley & Sons (1998).
26. Festinger, L., *A Theory of Cognitive Dissonance*. Stanford: Stanford University Press (1957).
27. McIntyre, P., Loyalty not enough. *Business Review Weekly*, 22, 15 December, 104-107 (2000).
28. Crosby, P.B., *Quality is Free*. New York: Penguin Books (1980).
29. Aranda, R.R., Fiddaman, T. and Oliva, R., Quality microworlds: modeling the impact of

quality initiatives over the software product life cycle. *American Programmer*, 6, **5**, 52-61 (1993).

30. Chichakly, K.J., The bifocal vantage point: managing software projects from a systems thinking perspective. *American Programmer*, 6, **5**, 18-25 (1993).
31. Abdel-Hamid, T.K. and Madnick, S.E., *Software Project Dynamics: An Integrated Approach*. Englewood Cliffs: Prentice Hall (1991).
32. Abdel-Hamid, T.K. and Madnick, S.E., Lessons learned from modeling the dynamics of software development. *Communications of the ACM*, 32, **12**, 1426-1455 (1989).
33. Abdel-Hamid, T.K., The dynamics of software project staffing: a system dynamics based simulation approach. *IEEE Transactions on Software Engng.*, 15, **2**, 308-318 (1989).
34. Forrester, J.W. and Senge, P.M., *Tests for Building Confidence in System Dynamics Models*. In: Legasto, A.A., Forrester, J.W. and Lyneis, J.M. (Eds), System Dynamics. New York: North Holland 209-228 (1980).
35. Carlson, B.R., *An Industrialist Views Industrial Dynamics*. In: Roberts, E.B. (Ed.), Managerial Applications of System Dynamics. Waltham: Pegasus Communications, 139-144 (1999).
36. Fey, W.R., *An Industrial Dynamics Case Study*. In: Roberts, E.B. (Ed.), Managerial Applications of System Dynamics. Waltham: Pegasus Communications, 117-138 (1999).
37. Schlager, K.J., *How Managers Use Industrial Dynamics*. In: Roberts, E.B. (Ed.), Managerial Applications of System Dynamics. Waltham: Pegasus Communications, 145-153 (1999).
38. Sterman, J.D., *Business Dynamics: Systems Thinking and Modelling for a Complex World*. New York: Irwin McGraw-Hill (2000).

## APPENDIX 1: THE LANGUAGE OF *iTHINK*

Essentially, *iThink* is a language that can be used to tell a story. System dynamics models described by it use the following elements of grammar to tell their story:

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 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.

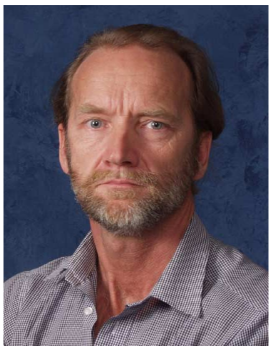
Converters, ○, can be thought of as adverbs that modify flows. They are often used to break out the detail of the logic, that might otherwise be buried within a flow, and might be used to represent constant values. These typically influence the behaviour of the regulators on the flows.

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.

## BIOGRAPHIES

Craig W. 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 software developer for Wesfarmers Ltd while studying towards a PhD in computer science at Edith Cowan University. His particular focus is on system dynamics and software engineering.



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 was previously Adjunct Professor in Computer Control Systems at the Technical University of Denmark. He is an internationally recognised authority in laboratory automation and has published a commissioned book in this field.