CE CONCURRENT ENGINEERING: Research and Applications

The Liar's Club: Concealing Rework in Concurrent Development

David N. Ford[1,*] and John D. Sterman[2]

[1]*Department of Civil Engineering, Texas A&M University, College Station, TX 77843-3136, USA*

[2]*Sloan School of Management, Massachusetts Institute of Technology, 50 Memorial Drive, E53-351, Cambridge, MA 02142 USA*

Abstract: Successfully implementing concurrent development has proven difficult for many organizations. However, many theories addressing concurrent development treat either technical aspects of the development process (e.g., precedence relationships) or behavioral issues (e.g., creating effective cross-functional teams), but not their linkages. We argue that much of the complexity of concurrent development—and the implementation failures that plague many organizations—arises from interactions between the technical and behavioral dimensions. We use a dynamic project model that explicitly represents these interactions to investigate how a ''Liar's Club''—concealing known rework requirements from managers and colleagues—can aggravate the ''90% syndrome,'' a common form of schedule failure, and disproportionately degrade schedule performance and project quality. We discuss the role of the incentives on and behavior of engineers and managers in concurrent development failure and explore policies to improve project performance.

Key Words: concurrent development, concurrent engineering, iteration, rework, cycle time, project management, concealment, system dynamics.

1. Introduction

Firms seeking competitive advantage to increase market share, profit, and growth have turned to concurrent development to speed the introduction of new products and beat their competitors to market. [17,24,26,29,40]. Despite some successes, implementing concurrent development has proven difficult for many organizations [25,40]. Implementation failure is not unique to concurrent development. Similar failures plague firms seeking to implement a wide range of process improvement tools such as TQM, reengineering, and diverse best practices in product development [27,36]. The hallmark of these failures is a mismatch between the technical, organizational, and dynamic complexity of the process and the mental models of the managers responsible for them, mental models that lead to inappropriate organizational structures, policies, and decisions. Studies of human decision making show that our mental models of complex systems are often simplistic and unreliable. Our ability to understand the unfolding impacts of our decisions is poor. Our mental models tend to have narrow boundaries and short time horizons: We find it difficult to incorporate interactions and feedbacks that cut across traditional functional, disciplinary, or academic boundaries. We take actions that make sense from our short-term and local perspectives, but, due to our imperfect appreciation of complexity, often feed back to hurt us in the long run [35].

Many explanations have been suggested for the concurrent development implementation challenge. Backhouse and Brookes [4] suggest implementation fails due to mismatches among a development organization's people, controls, tools, processes, and structure, and the organization's need for efficiency, focus, incremental change, radical innovation, and proficiency. Other researchers focus on organizational issues [5], personnel selection [14,33], personnel characteristics [20], and information transfer [9,21].

Unfortunately, many existing theories treat either the structure of development processes without regard to behavioral issues (e.g., [2,11,18,22]) or focus on development team or participant characteristics without considering the process structure (e.g., [14,20]). We argue that improving the effectiveness of concurrent development requires models that explicitly account for interactions and feedbacks among technical, organizational, and behavioral features of the development process. In this paper we develop a formal dynamic model to explore interactions of concurrent process structure with the behavioral decision processes of the actors in the system. We show how technical aspects of

*Author to whom correspondence should be addressed.
E-mail: DavidFord@tamu.edu

1063-293X/03/03 0211–9 $10.00/0 DOI: 10.1177/106329303038028

the development process such as overlapping activities, activity durations, and delays in the discovery of rework requirements interact with the behavior of developers and their managers to create unplanned iteration, delays, higher costs, and lower quality. We explore policies that can help improve project performance.

2. The 90% Syndrome

The "90% syndrome" is a common concurrent development problem in which a project reaches about 90% completion according to the original schedule but then stalls, finally finishing after about twice the original project duration has elapsed. The syndrome is common in industries including software, construction, consumer electronics, and semiconductors [1,10,19]. Our fieldwork with a leading semiconductor maker provides a typical example. Figure 1 contrasts the planned and actual progress of ASIC (Application Specific Integrated Circuit) development projects we call Python (top) and Rattlesnake (bottom). Python remained close to the original schedule through week 20 and was 73% complete by the original deadline. Progress then slowed from 1.8% per week to 0.9% per week, and the project was ultimately completed 77% late (week 69 vs. 39). Rattlesnake was worse: Reported progress rises to 79% by the original deadline (week 34) but two unplanned iterations delay completion until over twice the original scheduled duration (81 vs. 34 weeks). Unplanned iterations and slow late-stage progress are typical of the 90% syndrome.

To investigate the interaction of physical and information processes with managerial decision-making we built a dynamic model of concurrent development processes, described in [13], in this issue. The full model is available at <ceprofs.tamu.edu/dford> in the Vensim simulation language (see <www.vensim.com>).

3. Concealing Rework Requirements in Concurrent Development

We argue in [13] that concurrent development not only increases the vulnerability of projects to changes and errors requiring rework, but also increases the fraction of work released that will require changes. Increased concurrency reduces the availability of final

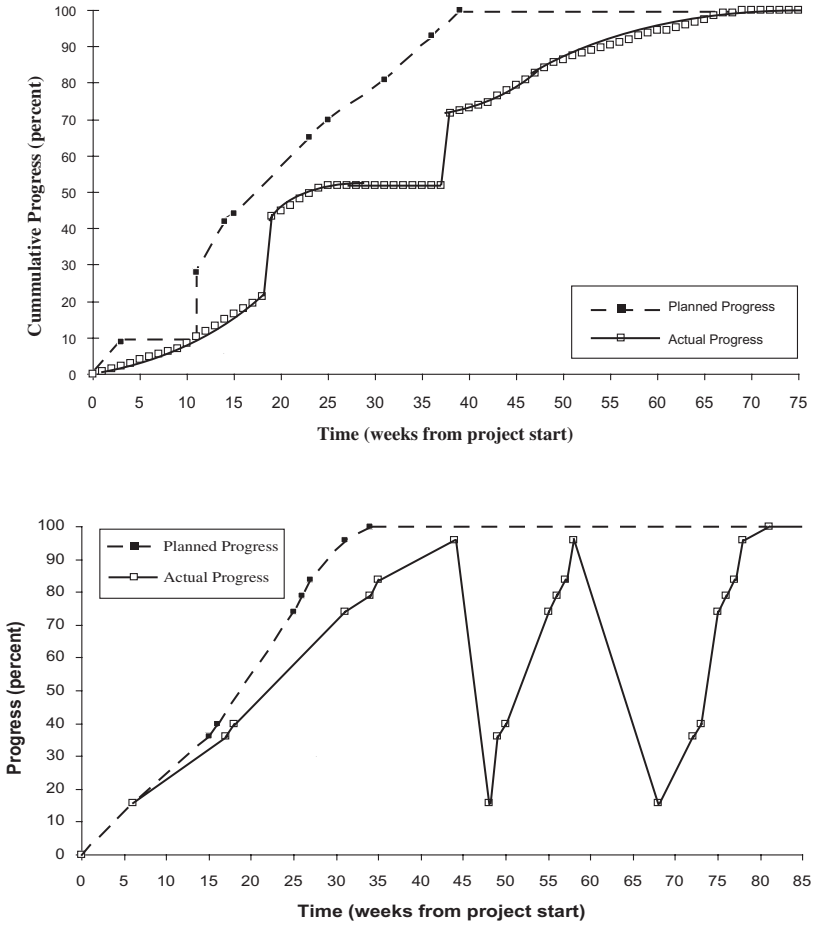

Figure 1. Planned and actual progress of the Python (top) and Rattlesnake (bottom) projects.

information, technologies, and components, leading to additional rework and iteration. In addition, behavioral effects arising from the increased pressure felt by developers to meet schedules that do not account for the greater iteration and coordination required by concurrency contribute significantly to the poor performance of many such projects.

Here we focus on one common and problematic behavior: purposefully concealing known problems from development team members and managers. As we describe next, concealing problems can have very different causes and impacts than the concealment of implementation details, such as in the modular design of software [37] or other products [6] or the decomposition of work [41]. Concealing known problems can be used to temporarily reduce work. For example, an engineer in a leading electronics company reported to us that design engineers regularly delayed revealing problems they discovered to avoid time-consuming document control work required by the organization's engineering change notice process [12]. The practice of concealing rework requirements is reinforced by people's dislike of bad news and information that contradicts their beliefs. People in authority often "shoot the messenger." Consequently, developers suppress information they believe will be unpleasant to their superiors or customers. For example suppliers are reticent to report that parts will be late even when they are members of their customer's development team [30]. The practice of hiding one's mistakes is institutionalized in many organizations. A manager at a major automobile manufacturer we will call AutoCo observed.

> "There is a basic cultural commandment in engineering—don't tell someone you have a problem unless you have the solution. You're supposed to solve it—and then tell them." [30, pp. 13–14]

Concealment is often standard practice. At a major defense contractor, weekly meetings of project team leaders were known as "the liars' club" because everyone withheld knowledge that their subsystem was behind schedule. Members of the liar's club hoped someone else would be forced to admit problems first, forcing the schedule to slip and letting them escape responsibility for their own tardiness. Everyone in the liar's club knew that everyone was concealing rework requirements and everyone knew that those best able to hide their problems could escape responsibility for the project failing to meet its targets.

Purposefully concealing changes from managers and other team members undermines core principles of concurrent development. The team concept of concurrent development is designed to promote open and early sharing of problems. But our fieldwork shows that this is often not practiced, even in organizations staffed with competent, well-intentioned developers and managers well-trained in concurrent development. What causes these managers to conceal change requirements? Concealing problems provides the manager of an individual phase several benefits beyond the avoidance of responsibility for poor performance cited above:

- Concealing known problems reduces the need for iteration (temporarily), increasing the amount of work management perceives to be complete and can make available to other activities. Apparent schedule performance improves, sometimes allowing managers to meet a critical deadline they would miss if change requirements were revealed.
- Concealment reduces the starvation of receiving phases and therefore avoids pressure from them to release faster. Often a receiving phase starved for work will complain to project leadership, leading to unwanted attention on the supplying phase, attention that may persist throughout the project and beyond, as the managers of that phase develop a bad reputation.
- Concealing known problems reduces the work acknowledged to need coordination and rework and therefore the amount of work considered a problem or hindrance to progress, improving apparent project quality.
- Delaying coordination and rework spreads the total effort required by a concealing phase over a longer period of time, thereby reducing peak resource requirements.
- Hiding rework requirements increases the chance that schedule extensions caused by other phases will provide cover for managers and engineers to resolve their own problems.
- Maintaining adequate apparent progress reduces the likelihood that upper management will intervene or replace the manager with someone believed to be better able to meet targets. Concealment enhances the manager's job security and authority.

We simulate the effects of concealing rework requirements by disaggregating the work known to require rework into a fraction acknowledged and a fraction concealed. Tasks found to require rework that is then concealed are approved and eventually released, instead of moving into the stock of tasks requiring coordination and rework. Concealment therefore acts like a drop in the effectiveness of quality assurance. Modeling concealment in this fashion does not change the probability that a task requires rework, increase project scope or complexity, nor reduce the quality of the work done (conditioned on the information available to the phase). We assume only that concealment results in the approval of some work that is known to require rework.

To calibrate the model to the Python project ([13] this issue) we estimated that the pressure to show progress

caused personnel in the requirements, design, and prototyping phases to conceal half of the rework requirements they discover. While 50% concealment may appear high, our fieldwork suggests this fraction is often exceeded. We also assume, a fortiori, that the final test phase detects all errors and does not conceal any of them. In reality, testing is less than 100% effective and managers of the test phase may conceal known problems when faced with deadline pressure. Note that testing cannot uncover certain types of errors made in the product definition phase, so that the product may not meet customer requirements even though it is fully compliant with the specifications released to the designers—a common situation.

Figure 2 simulates the Python project assuming 0 and 100% concealment of rework requirements in the product definition, design, and prototyping phases. Concealment shows faster progress through most of the original 39 week schedule, but delays the last portion of the project into the characteristic pattern of the 90% syndrome (and even a 10 week period where the perceived fraction complete falls as testing reveals some of the errors other phases did not report). The project is completed 98% (38 weeks) later than originally planned and 15 weeks (25%) later than when managed without concealment. Concealment increases total work effort as later discovery of errors means more work has to be redone when the errors are found (from 134% of project scope with no concealment to 200% of project scope under full concealment). Concealment also dramatically reduces project quality by causing more errors in product definition to remain undiscovered when the project is completed (errors released rise from less than 1% of project scope with no concealment to 6.8% under full concealment). Concealment is locally rational for individuals but globally irrational for the organization.

Figure 3 shows that increasing concealment significantly lengthens project durations, increases costs (through more iteration and rework), and lowers quality (more errors are released with the product). The results also show an interaction between concurrence and concealment: The greater the concealment of known problems, the smaller is the cycle time reduction enabled by increasing concurrence, and the lower the quality of the final product. For example, increasing concurrence by 50% relative to the base case reduces project duration almost 30 weeks when there is no concealment, but only 11 weeks when there is 100% concealment. Similarly, defects remaining when the project is released to the customer rise 0.4% from 1 to 1.4% of project scope when there is no concealment (a 40% increase), but rises 3.0% from 6.8 to 9.8% of project scope when the concealment fraction is 1.0 (a larger increase of 44% on a much larger base). Concealment has greater impact with high concurrence because more work has been done before the errors that are concealed are finally detected and revealed, and because greater concurrence generates more errors and thus causes more tasks requiring rework to be concealed.

It is tempting to argue that managers with better training or more experience with concurrent development would resist the temptation to conceal problems and join a liar's club. But consider the predicament of managers of individual phases. Aggressive deadlines and overly optimistic assumptions about productivity and work quality mean most soon find themselves behind schedule. The pressure can be enormous. A manager at a major automobile maker said "... the only thing they shoot you for is missing product launch... everything else is negotiable." [28]. Managers have several options once they realize that revealing problems would cause them to miss a deadline. They can reveal the problem, miss their original deadline, and perhaps be identified as the manager responsible for delaying the entire project—thus risking being "shot" for "missing product launch". Alternatively, they can identify the errors they inherited from other phases but conceal the problems generated by their own phase, blaming their peers for problems their phase is experiencing. If the deadline then slips, the managers have gained time to solve the problems generated by their own phase without having to take responsibility for them. But they will alienate the managers of the phases

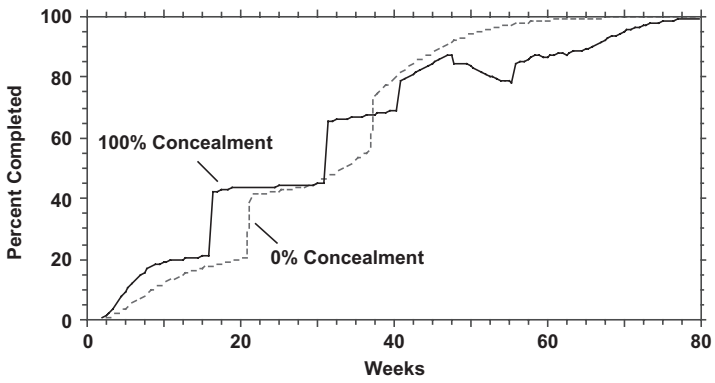

Figure 2. Simulated Python project with different levels of rework concealment.

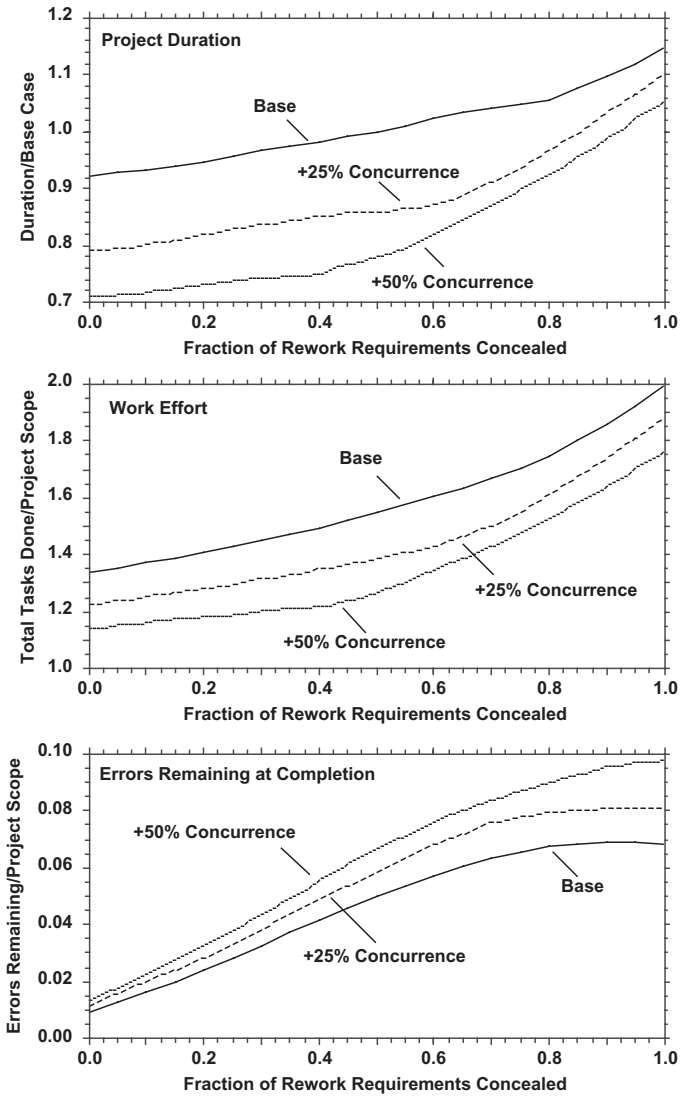

Figure 3. Impact of concealment on duration, work effort, and quality.

they exposed and open the door to retaliation. Only by joining the liar's club and remaining silent about both their own problems and those generated by other phases can they avoid responsibility for failure, prevent retaliation by other managers, and seek to solve their problems in relative secrecy.

The liar's club forms a prisoner's dilemma (PD) in which engineers play the role of prisoners being interrogated about their progress by management. The engineers can cooperate with one another by concealing the problems they know to exist, or defect by revealing those problems to management. If all engineering groups cooperate by concealing known problems the schedule remains the same and they avoid blame for delays (a small payoff). If they reveal problems caused by others they get a large payoff because the schedule will slip, giving them a chance to fix their own problems without detection; their colleagues then face a large negative payoff as management blames them for the delay. However, if all engineering groups defect by revealing all known problems, the schedule slips, but everyone faces the wrath of management, a large negative payoff for all.

In most literature on social dilemmas like the PD cooperation is seen as desirable, and the challenge has been to find conditions that encourage greater cooperation, raising total welfare. Here, however, cooperation hurts overall project performance, and the policy goal is to promote defection (that is, to promote an atmosphere of honesty and early disclosure). An extensive literature (beginning with Axelrod [3]) shows that cooperation can flourish in the repeated PD, where participants choose to cooperate or defect multiple times. The more people expect to interact, and the more they expect to interact with the same people, the greater the chance for cooperation. Development projects constitute a repeated PD where the engineers work with the same people throughout a project, and have the chance to

cooperate or defect every day. Even after the current project ends, engineers expect to be assigned to new projects in which they will again face the choice of cooperation or defection, and will be working with many of the same people. These conditions strongly favor concealment of known problems. Given the risks and benefits of concealment described above, it is small wonder that many development organizations have developed strong liar's clubs with a self-replicating culture of concealment.

The liar's club can be strongly self-reinforcing even though it is dysfunctional for the organization [28]. In the short run it is rational for individual managers to conceal change requirements, especially if they believe other phases also do so. However, concealment worsens overall progress for the entire project by preventing "... synchronization of information exchange, decisions, and iterations across processes" recommended by Browning [9]. The resulting delays, cost overruns, and quality problems may feed back to intensify financial stress on the organization, leading to even more aggressive schedules and greater pressure to conceal changes in future projects. Managers who conceal successfully are likely to be rewarded and promoted, teaching personnel throughout the firm that concealment is the route to career success.

There are some counteracting forces that might weaken the self-enforcing dynamic of widespread concealment. Members of the liar's club may try to reveal the flaws of other phases anonymously to gain time and shift blame but escape retaliation. An AutoCo developer reported that "... the supplier will say things like (for example), 'You didn't hear it from me, but something is going to be late and somebody's lying to you. Don't tell anyone I told you.'" ([30], pp. 18–19). Leaking, however, eventually results in tighter controls over information and still less communication among phases. One might hope that successful engineers, once promoted to management, would know about the liar's club and take steps to counter it. Instead, as documented in Repenning and Sterman [28], senior project personnel sometimes respond by further accelerating schedules and increasing the pressure on teams to finish as fast as possible, as illustrated by a manager in an automobile maker, who recalled

> "... [one] executive engineer used to have what I would call 'fighting meetings'.... His attitude was we should tell everybody that they're all [behind schedule], they're all failing, they have to make changes faster [and that] if we don't make changes right now, we're going to shut the whole division down..."

The consequence, of course, is to create even stronger incentives for concealment.

4. Discussion: Integrated Iteration Design and Management

The frustration of competent and well-intentioned managers in concurrent development projects can be understood as resulting from process-constrained progress, magnified by concurrent development practices, and distorted by short-sighted management policies. Shifting development focus and concealing rework requirements in response to the schedule pressure induced by concurrent development practices shifts the burden of and responsibility for change discovery away from individual phases while temporarily improving apparent performance. Such behavior creates the impression that projects are proceeding as planned throughout most of the original schedule, so managers do not receive signals that could initiate corrective action. By the time changes are discovered and acknowledged, project managers face a large backlog of hidden rework. The resulting iteration cycles delay completion and increase cost even when resources are ample because the iterations are constrained by the underlying concurrent development structure of information exchange. The liar's club and incentives for individual phases to conceal known problems suggests that solving the "how frequent to meet" problem addressed by Ha and Porteus [15] or "when to meet" problem addressed by Loch and Terwiesch [22] can be inadequate. Since managers typically have less influence over processes than resources, they have few effective tools and methods with which to accelerate throughput when rework discovery delays and iteration cycles constrain progress. Indeed, attempts to remedy these delays through overtime and hiring can worsen the problem through fatigue, skill dilution, and higher coordination and training requirements [8,16,39]. The interaction between process and behavior helps explain how the 90% syndrome can occur in stable, competently managed, adequately staffed projects and organizations.

Effective policies must integrate the decision-making heuristics people actually use with the physical structure and information flows characterizing a project. For example, changing the perceived payoffs of members of the liar's club to reward the early disclosure of problems and (more difficult) the cultural norms that support the club could change both managerial behaviors and the paths and speeds of information flows; these benefits would then make increased concurrence more effective. However research that integrates physical processes and behavior are rare. Most studies addressing team characteristics, worker motivation and skill, and other human resource issues generally omit consideration of the process structure, and vice-versa. For example, Moffatt [25] concludes that "... team effectiveness [in goal setting, conflict resolution, and decision-making] and project schedule simultaneity [process concurrence]

are separate, independent paths to improved project performance.'' We argue instead that the structure of concurrence relationships and information flow interact strongly with the decision-making behavior of the people working in and managing the project. Scholars and practitioners need to integrate the technical dimensions of project management (e.g. critical path/PERT, precedence relationships, Design Structure Matrix) with the behavioral aspects (e.g. the liar's club, culture, incentives) to identify high leverage policies for improved project management.

Changing the perspectives of managers and developers—from optimizing phase performance to optimizing project performance, from a focus on technical *or* behavioral issues to the integration of technical *and* behavioral issues—faces serious implementation barriers. The bounded rationality of managers limits the scope and complexity they can perceive, understand and use individually [32] or in teams [7]. Many development projects may be too big and complex for managers to simultaneously ''think globally'' (at the project level) but ''act locally'' (operate in specific phases) because it requires both the expansion of their mental models and aligning their local incentives with the global goals of the project. Our fieldwork indicates that when forced to choose, most managers focus locally where they can have both an understanding of the impacts of policies and the influence to implement those policies.

Effective strategies address the managerial behaviors that cause iteration cycles to constrain progress. Various management and project team member decisions, such as concealing known problems, can cause excess unplanned iteration. Each iteration introduces delay, but in addition these unplanned iterations increase the distance information must travel, slow the speed at which information traverses the distance, and occur later than they might have. All these characteristics lead to disproportionate increases in costs and loss of quality by increasing the amount of work done in good faith that must be scrapped, forcing management to make more and larger unplanned changes in resource allocation, decreasing skill and experience of workers assigned to a project, increasing the likelihood of excessive overtime that leads to worker burnout and higher error rates, and so on. Researchers have proposed process designs to manage iteration cycle number, speed, length, or timing. For example Terwiesch et al. [38] recommend ''a fast process of problem detection, problem solving, and engineering change implementation'' which increases iteration cycle speed. They suggest ''loosening the coupling (dependence) between development activities'' and improving the accuracy of preliminary information, both which reduce the number of cycles. McAllister and Backhouse [23] suggest redesigning work flows to reduce the number of iteration paths in a project network. There may be ways to partition development to reduce interactions among subsystems, to improve the accuracy of preliminary information, and to minimize the number and length of iterations, but doing so requires deep knowledge of the interactions and of customer requirements, and the organizational and human resource flexibility to act. Developing such knowledge and flexibility takes time and effort, resources that are already scarce in most organizations. It is far easier and more tempting to ignore interactions, release specifications and subsystems before they are mature, and conceal the need for rework until forced by circumstances to reveal them.

Process changes cannot improve concurrent development project performance if they do not also address the behaviors that drive iteration cycles such as the policy of concealing rework requirements. Information technology can play a role by acting on both processes and behaviors simultaneously. For example Sabbagh [31] reports Boeing's use of computer assisted design and drafting (CADD) to identify conflicts among functional systems for use of the limited space in the aircraft, reducing the potential for concealing change requirements and accelerating error detection. Such virtual worlds and simulations offer great potential for shortening the delay in discovering rework requirements and preventing concealment. At the same time, creating such virtual worlds is challenging—to be technically sound the models must include interactions that cross established disciplinary and functional boundaries. To be used, project participants must have confidence that the models are not biased against their particular specialty or phase. Creating the needed trust and aligning incentives takes time and effort, resources that are often scarce. Yet it can be done, as shown by examples such as those described in [34 (Ch. 2, 6.3.4)]; see also [28].

In this paper we have used a dynamic model of a development project to describe, quantify and simulate how physical and information processes interact with managerial decision making to constrain progress and cause project overruns. We have shown how concealing known rework requirements is locally rational but globally irrational. Such concealment is common in development projects. It arises from the mental models of the managers and developers of individual phases and is amplified by concurrent processes, local incentives, and organizational cultures favoring concealment. Our model can illustrate these interactions to practitioners and facilitate improvements in project design and management.

Acknowledgment

The authors thanks the Organizational Learning Center and the System Dynamics Group at the MIT Sloan School of Management and the Python

organization for financial support. Special thanks to the members of the Python project for their interest, commitment, and time.

References

1. Abdel-Hamid, T. (1988). Understanding the "90% Syndrome" in Software Project Management: A Simulation-Based Case Study, *Journal of Systems and Software*, **8**: 319–330.
2. Adler, P.S., Mandelbaum, A., Vien, N. and Schwerer, E. (1995). From Project to Process Management: An Empirically-based Framework for Analyzing Product Development Time, *Management Science*, **41**(3): 458–484.
3. Axelrod, R. (1984). *The Evolution of Cooperation*, New York: Basic Books.
4. Backhouse, C.J. and Brookes, N.J. (1996). *Concurrent Engineering, What's Working Where*, Gower, Brookfield, VT: The Design Council.
5. Bailey, D.D. (Nov. 1999). Challenges of Integration in Semiconductor Manufacturing Firms, *IEEE Transactions of Engineering Management*, **46**(4): 417–428.
6. Baldwin, C.Y. and Clark, K.B. (2000). *Design Rules: The Power of Modularity*, MIT Press, Cambridge, MA.
7. Brehmer, B. (1998). Effects of Time Pressure on Fully Connected and Hierarchical Architectures of Distributed Decision-making, In: Yvonne Waern (ed.), *Co-operative Process Management, Cognition and Information Technology*, London: Taylor & Francis, Ltd.
8. Brooks, F.P. (1978). *The Mythical Man-Month*, Reading, MA: Addison-Wesley.
9. Browning, T. (Oct. 1999). Sources of Schedule Risk in Complex System Development, *Systems Engineering*, **2**(3): 129–142.
10. DeMarco, T. (1982). *Controlling Software Projects*, Yourdon, New York.
11. Ettlie, J.E. (1995). Product-Process Development Integration in Manufacturing, *Management Science*, **41**: 1224–1237.
12. Ford, D.N., Hou, A. and Seville, D. (1993). An Exploration of Systems Product Development at Gadget Inc. System Dynamics Group Report D-4460, Sloan School of Management. Massachusetts Institute of Technology, Cambridge, MA.
13. Ford, D.N. and Sterman, J.D. (2003). Overcoming the 90% Syndrome: Iteration Management in Concurrent Development Projects, *Concurrent Engineering: Research and Applications* (this issue).
14. Gerwin, D. and Moffat, L. (1997). Withdrawl of Team Autonomy During Concurrent Engineering, *Management Science*, **43**(9): 1275–1287.
15. Ha, A.Y. and Porteus, E.L. (1995). Optimal Timing of Review in Concurrent Desing for Manufacturabillity, *Management Science*, **41**(9): 1431–1447.
16. Haddad, C. (1996). Operationalizing the Concept of Concurrent Engineering: A Case Study from the US Auto Industry, *IEEE Transactions on Engineering Management*, **43**(2): 124–132.
17. Hayes, R.H., Wheelwright, S.C. and Clark, K.B. (1988). *Dynamic Manufacturing, Creating the Learning Organization*, New York: The Free Press.
18. Joglekar, N.R., Yassine, A.A., Eppinger, S.D. and Whitney, D.E. (2001). Performance of Coupled Development Activities with a Deadline, *Management Science*, **47**(12): 1605–1620.
19. Kiewel, B. (January 1998). Measuring Progress in Software Development, *PM Network*, Project Management Institute, **12**(1): 29–32.
20. King, N. and Majchrzak, A. (1996). Concurrent Engineering Tools: Are the Human Issues Being Ignored, *IEEE Transactions on Engineering Management*, **43**(2): 189–201.
21. Krishnan, V. (1996). Managing the Simultaneous Execution of Coupled Phases in Concurrent Product Development, *IEEE Transactions on Engineering Management*, **43**(2): 210–217.
22. Loch, C.H. and Terwiesch, C. (1998). Communication and Uncertainty in Concurrent Engineering, *Management Science*, **44**(8): 1032–1048.
23. McAllister, J. and Backhouse, C. (1996). An Evolving Product Introduction Process. In: Backhouse, C. and Brookes, N. (eds), *Concurrent Engineering, What Works Where*, Gower, Brookfield, VT.
24. Meyer, C. (1993). *Fast Cycle Time, How to Align Purpose, Strategy, and Structure for Speed*, New York: The Free Press.
25. Moffat, L.K. (1998). Tools and Teams: Competing Models of Integrated Product Development Project Performance, *Journal of Engineering Technology and Management*, **15**: 55–85.
26. Patterson, M.L. (1993). *Accelerating Innovation, Improving the Process of Product Development*, New York: Van Nostrand Reinhold.
27. Repenning, N.P. and Sterman, J.D. (2000). Getting Quality the Old-Fashioned Way: Self-Confirming Attributions in the Dynamics of Process Improvement, In: Scott, R. and Cole, R. (eds), *Improving Theory and Research on Quality Enhancement in Organizations*, pp. 201–235, Thousand Oaks, CA: Sage.
28. Repenning, N. and Sterman, J. D. (2001). Nobody Ever Gets Credit for Fixing Defects that Didn't Happen: Creating and Sustaining Process Improvement, *California Management Review*, **43**(4): 64–88.
29. Rosenthal, S.R. (1992). *Effective Product Design and Development*, Homewood, IL: Business One Irwin.
30. Roth, G. and Kleiner, A. (1996). The Learning Initiative at the AutoCo Epsilon Program, 1991–1994, Center for Organizational Learning, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA.
31. Sabbagh, K. (1995). *21st Century Jet*, *The Making of the Boeing 777*, London: Pan Books.
32. Simon, H.A. (1995). *The Sciences of the Artificial*, Cambridge, MA: The MIT Press.
33. Stahl, J., Luczak, H., Langen, R., Weck, M., Klonaris, P. and Pfeifer, T. (1997). Concurrent Engineering of Work and Production Systems, *European Journal of Operational Research*, **100**: 379–398.
34. Sterman, J.S. (2000). *Business Dynamics, Systems Thinking and Modeling for a Complex World*, New York: Irwin McGraw-Hill.
35. Sterman, J.D. (1994). Learning in and about Complex Systems, *System Dynamics Review*, **10**(2–3): 291–330.

36. Sterman, J., Repenning, N. and Kofman, F. (1997). Unanticipated Side Effects of Successful Quality Programs: Exploring a Paradox of Organizational Improvement, *Management Science*, **43**(4): 503–521.
37. Sullivan, K.J., Griswold, W.G. and Ben Hallen, Y.C. (September 2001). The Structure and Value of Modularity in Software Design, In: *Proceedings of the Joint International Conference on Software Engineering and ACM SIGSOFT Symposium on the Foundations of Software Engineeering*, Vienna.
38. Terwiesch, C., Loch, C.H. and DeMeyer, A. (1998). Preliminary Information, Interdependence and Task Concurrency in Product Development, Forthcoming in Organization Science.
39. Thomas, H.R. and Raynar, K.A. (1994). Effects of Scheduled Overtime on Labor Productivity: Quantitative Analysis, Source Document 98, Construction Industry Institute, Austin, TX.
40. Wheelwright, S.C. and Clark, K.B. (1992). *Revolutionizing Product Development, Quantum Leaps in Speed, Efficiency, and Quality*, New York: The Free Press.
41. Yassine, A., Joglekar, N., Braha, D., Eppinger, S. and Whitney, D. (2003). Information Hiding in Product Development: The Design Churn Effect, Forthcoming Research in Engineering Design, Volume 14.