Overcoming the 90% Syndrome: Iteration Management in Concurrent Development Projects

David N. Ford and John D. Sterman

Concurrent Engineering 2003; 11; 177
DOI: 10.1177/106329303038031

CONCURRENT ENGINEERING: Research and Applications

Overcoming the 90% Syndrome: Iteration Management in Concurrent Development Projects

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 to reduce cycle time has proven difficult due to unanticipated iterations. We develop a dynamic project model that explicitly models these interactions to investigate the causes of the "90% syndrome," a common form of schedule failure in concurrent development. We find that increasing concurrence and common managerial responses to schedule pressure aggravate the syndrome and degrade schedule performance and project quality. We show how understanding of and policies to avoid the 90% syndrome require integration of the technical attributes of the project, the flows of information among participants, and the behavioral decision-making heuristics participants use to respond to unanticipated problems and perturbations.

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

1. Introduction

Developing products faster has become critical to success in many industries, whether the product is an office building, software package, or computer chip. Calls for faster product development have simultaneously taken on the sacred tone of a mantra and the volume of a brass band [34]. Perhaps with good reason. Cycle time reduction is considered crucial to success by many researchers (e.g., [34,41]) and practitioners (e.g., [29,33]). Developing products faster than competitors can increase market share, profit, and long-term competitive advantage [29,33,41].

In response many firms have shifted from sequential to concurrent development (aka Integrated Product Development or Fast Track development). Large reductions in cycle time can be realized by applying concurrent development [4,9,31,41,42]. Despite some successes, implementing concurrent development has proven difficult for many [30,41]. These failures arise in part because cycle time reduction through concurrent development increases process and organizational complexity [8,26,41]. Concurrent methods often increase the frequency and number of information transfers among project phases [7,8]. More tasks are begun with incomplete or preliminary information, increasing iteration. Management policies have not generally improved to address the effects of increased complexity created by concurrent development.

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 the disaggregation of work into smaller pieces [35,43] and mismatches between attributes of the technology and the degree of overlapping employed [25]. Still others focus on activity sequencing [22,35], coordination caused by overlapping activities [21], and information transfer [6,25].

In this paper we develop a formal model to explore how concurrent process structures can cause a particular form of development project schedule failure, the 90% syndrome. We show how development processes such as overlapping of activities, activity durations, and delays in the discovery of rework requirements can create unplanned iteration, delays, higher costs, and lower quality. We explore policies that can help improve project performance. In the companion paper ([19], this issue), we use the model to explore the interactions between the process structure of concurrent projects and behavioral responses of developers and managers.

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

1063-293X/03/03 0177–10 $10.00/0 DOI: 10.1177/106329303038031

2. The 90% Syndrome

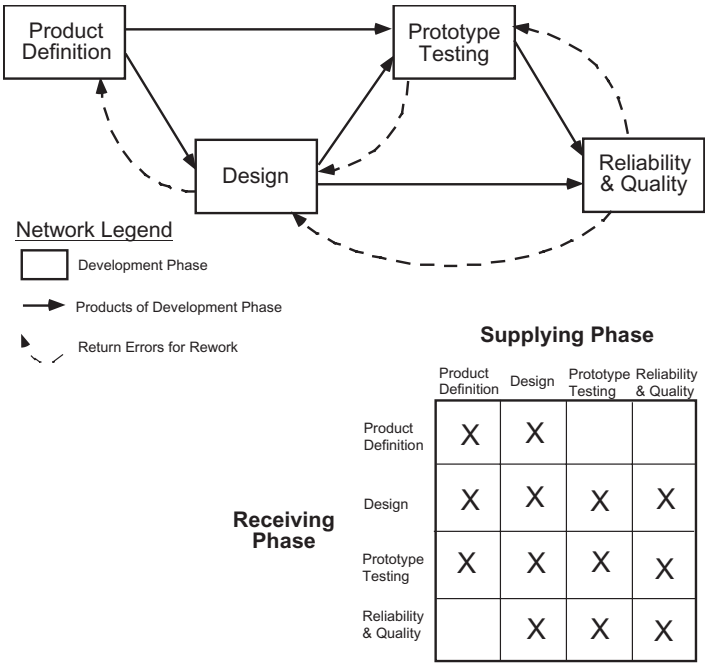
One common concurrent development problem is the "90% syndrome." The syndrome describes a project that reaches about 90% completion on schedule but then stalls, finally finishing after about twice the originally projected duration. A senior manager in one company we worked with described their experience as "The average time to develop a product is 225% of the projected time, with a standard deviation of 85%. We can tell you how long it will take, plus or minus a project" [16]. The syndrome is common in many industries including software, construction, consumer electronics, and semiconductors [2,12,24]. Consider the following example from our fieldwork with a leading semiconductor firm, describing an ASIC (Application Specific Integrated Circuit) project we call Rattlesnake. The original schedule called for a 34 week project, with a smooth, single-pass flow of work through product definition, design, layout, mask preparation, prototype fabrication, prototype testing, manufacturing process design, and production rollout—no iterations were anticipated. The project appeared to progress smoothly, though somewhat more slowly than planned, apparently completing 79% of the project scope by the original deadline. However, prototype testing revealed major problems, requiring an unplanned iteration with revisions in the design. Tests of the second prototype found still more problems, requiring another major iteration. The project was finally completed in week 81, more than twice the original schedule (see Figure 1 in [19], this issue). The slow progress experienced late in the project is typical of the 90% syndrome, and the unplanned inter-phase iterations suggest the importance of the late discovery of unanticipated rework.

3. The Development Project Model

To investigate the impacts of the interaction of physical and information processes with managerial decision-making we built a dynamic project simulation model that integrates several constraints usually treated separately:

1. *Characteristic Process Durations*: Development activities require minimum times to be performed regardless of the resources allocated to the activity.
2. *Development Activity Sequencing:* Development activities occur in a specific sequence within phases.
3. *Dynamic Information Requirements*: Development phases are constrained by information dependencies within and among project phases. These dependencies vary with phase progress and management decisions.
4. *Work Release Policies:* Work is often not released as it is completed, but in discrete packets. Policies governing packet size and release timing strongly affect the availability of information across project phases.
5. *Coordination*: When released work is discovered to require rework developers in the originating and discovering phases must coordinate prior to revising the work.

Our model simulates the performance of a multiple-phase development project. Each phase is represented by



Receiving Phase \ Supplying Phase	Product Definition	Design	Prototype Testing	Reliability & Quality
Product Definition	X	X		
Design	X	X	X	X
Prototype Testing	X	X	X	X
Reliability & Quality		X	X	X

Figure 1. A project phase network and its corresponding Design Structure Matrix.

a generic structure, which is parameterized to reflect a specific stage of product development such as preparing construction drawings, writing software, or testing prototypes. The unit of measure for development work is the "task" or work package, an atomic piece of work. Examples include writing a line of code or installing a steel beam. When tasks within a phase are heterogeneous the unit of work can be defined as the average amount an experienced person can accomplish in a given interval (e.g., an hour or day). Information concerning the quality of completed tasks is generated through testing or quality assurance efforts. Tasks may require rework because they were done incorrectly or because the work or information they were based on was itself erroneous or has changed.

The model is a system of nonlinear differential equations. The model is based on existing product development theories and our field studies of development projects. For example, the development process structure is based on theories of project constraints and resources [32] and previous dynamic project models including [1,10,17]. Because no closed-form solutions are known, we simulate the system's behavior. The full model is available at <ceprofs.tamu.edu/dford> in the Vensim simulation language (see <www.vensim.com>).

3.1 Modeling Work and Information Flows

The flow of work and information among phases defines the network structure of the project. Figure 1 shows a simple but common example. The links shown in Figure 1 represent several forms of interphase interaction, including:

- *Work progress* in which supplying phases provide development products or other information to receiving phases. These flows are shown by the solid arrows.
- *Work inherited* by receiving phases from supplying phases may require rework (either mandatory due to errors or optional for improvement). Inheriting work containing errors or requiring rework corrupts work done by the receiving phases. When corrupted work is discovered it is reported to the phase responsible for the problem so it can be reworked. These information flows are shown by the dashed arrows in the project network.
- *Rework* requires coordination between the phase that discovered the change requirement and the phase that generated and must correct the work. Coordination must occur prior to the revision of the work. Coordination is an activity in individual phases (boxes) generated by the reporting of problems requiring rework (dashed arrows).

The information flows among phases in the model are bi-directional, as in the Design Structure Matrix (DSM) approach [13,35,36]. The DSM identifies bi-directional dependencies between phases in which the activity initiated first (upstream) receives products from the activity initiated later (downstream) as well as the more traditional dependence in the general direction of work flow. Figure 1 also shows the DSM corresponding to the project network in the diagram. Several iterative loops are created by the bi-directional dependencies among phases. Our model allows any DSM to be represented by specifying the number of phases and the dependencies among them.

All development processes are constrained by the physical and information relationships among the activities and phases within a project. These constraints include development activity durations and precedence relationships, information dependencies leading to iteration [36], the availability of work [17], coordination mechanisms [22], the characteristics of information transferred among development phases [25], and the number, skill, and experience of project staff [2]. These processes and policies can interact to constrain progress. Consider the erection of the structural steel skeleton for a ten story building. Each member (the columns, beams, and bracing) must be installed, inspected, and corrected if the installation is found to be defective. These activities can only occur in a specific order: install, inspect, approve or discover a problem, rework, and re-inspect; when no further problems are found the work is approved and released so other work dependent on that task can proceed (e.g., installation of floors, walls, etc.). Management policies such as the number of floors that must be approved prior to release also affect progress and information availability downstream. Figure 2 shows the states of work within a single development phase in the model and the flows of work among them.

As work is first completed it enters the stock of work awaiting quality assurance (QA). If it passes QA (either because it is correct or the need for changes is not detected), it is approved and enters the stock of approved work. When sufficient work has been approved, a package is released, adding to the stock of work released to other phases or customers. The release package size is a management decision and is conditioned by characteristics of the phase. For example, in semiconductor development the vast majority of the design code must be completed prior to release for a prototype build since almost all the code is needed to design the masks. In other development settings managers have broad discretion in setting release package sizes.

Work found to require changes must be resolved through coordination with the phase responsible for the problem. Classic examples include designers working with marketers to refine ambiguous product

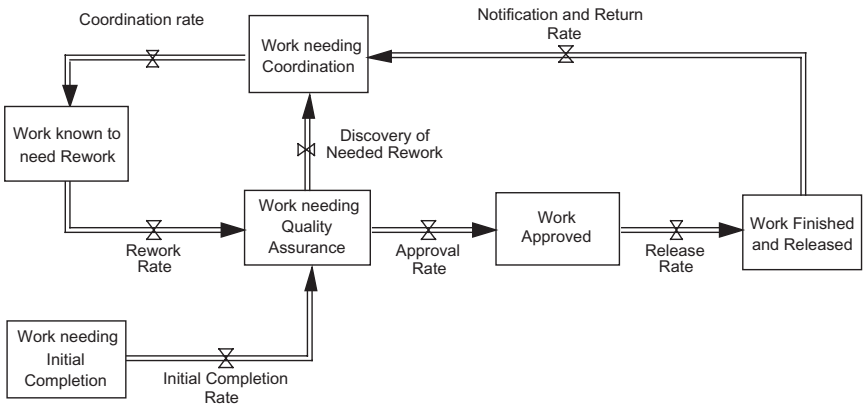


Figure 2. Work flows within a single development phase.

specifications and manufacturing engineers meeting with designers to explain why parts cannot be built as specified in the drawings. After coordination resolves disputed issues, these tasks move to the stock of work to be changed and are subsequently reworked, then returned to quality assurance for re-inspection. Quality assurance is imperfect, so some tasks requiring rework can be missed and are erroneously approved and released. Such rework requirements may be discovered later by another phase—if not, they remain embedded in the product, to be discovered by the customer. When the phase reports problems, the affected tasks are moved from the stock of work considered finished to the coordination backlog, then eventually reworked. For example, a test phase may discover a short circuit across two layers in a prototype chip. If the error is traced to the design, test engineers must work with the designers to specify the location and characteristics of the short circuit. The designers then must rework, recheck and rerelease the design, followed by layout, masking, prototype fabrication, and retesting of the new chip.

The probability of error detection depends on a variety of behavioral factors and management decisions. The probability of detecting problems declines as the information available is less current, complete, and accurate. High overlap between dependent phases in concurrent development means many tasks are done on the basis of specifications and components that are unavailable or changing. Our fieldwork shows that activities such as quality assurance, testing, documentation, and validation commonly suffer under concurrent development as development activities are overlapped. For example, we asked engineers in a large manufacturing firm how they accommodated the schedule compression and concurrency created by the organization's official product development process:

> "The technology may not be ready before alpha phase. Sometimes we have no choice, we just have to put something in" — Development Engineer

> "We might accept a lower level of maturity [in the prototype]. Maybe maturity isn't a good word. We might accept a lower level of design representativeness than we would like" — Engineering Manager

> "Often, we'll put [early prototype] parts in a [later prototype]... There's no getting around it as long as we have to go fast" — Program Manager

Schedule compression also biases workers towards getting their tasks done, even when that means spending less time on validation and quality assurance. An engineer in the organization above admitted that "We might not be able to finish the part, finish the FMEA's [Failure Modes and Effects Analysis], etc. We'll do the FMEA's, but they won't be as thorough as they would [be] otherwise." Another commented: "We haven't done the FMEA yet. We'll probably do it for beta [second prototype], but not for alpha [first prototype]. We just don't have time to do it."

These quotes illustrate a phenomenon we find repeatedly in our fieldwork: The greater the degree of concurrence, the greater the schedule pressure, and the greater the gap between resources available and resources required, the less effort is devoted to quality assurance and the lower the effectiveness of that effort. We capture these effects parsimoniously by assuming the probability of detecting the need for rework declines as the degree of concurrence increases.

3.2 Modeling the Speed of Development Activities and Concurrence

The rates at which development activities are performed depend on two types of constraints: resources and processes. Obviously, progress can be constrained by inadequate resources—too few workers, insufficient worker skill and experience, or insufficient supporting infrastructure (such as CAD/CAM systems). A variety of models explore how underestimating project scope, overestimating productivity, delays in the discovery of errors, or unexpected changes in customer requirements

can cause resource shortages that lead to delays, cost overruns, and quality problems (e.g., [2,10,37 Ch. 2.3]). In this paper, however, we seek to show how the structure of a development process interacts with managerial decision making to contribute to the 90% syndrome, even when resources are ample. If so, throwing more people and money at a project in trouble will have low leverage; effective policies will require changes in project structure and management policies.

Even when resources are ample, progress can be constrained by the interdependencies among phases and tasks. As an example, consider again the erection of the steel skeleton for a building. Each steel member must be installed (base work), and inspected (quality assurance). If an error is found, the affected supervisors and skilled trades must work together to devise a plan to remedy it (coordination) before the error can be corrected (rework). For any given technology, a certain minimum amount of time is required for each of these activities even when resources such as laborers and cranes are ample. Further, certain tasks cannot be started or completed until others are done. For example, the steel members for the upper floors cannot be installed until the beams and girders for lower floors are in place. These constraints are captured in our model through concurrence relationships. The function relating how much steel for upper floors can be placed to the progress of lower floors defines an *intraphase* concurrency relationship (the constraint arises within the steel erection phase).

Analogously, *interphase* concurrence relationships answer the question "How much work can we now complete given the work released by the phases upon which we depend?" For example, the erection of the steel for an office building depends on the release of construction drawings by the design phase and the progress of foundation work (among others). These constraints require two interphase concurrence relationships: one describing how much of the steel can be erected based on the release of construction drawings, and another describing how much steel erection can proceed based on the state of the foundations. Either of these interphase relationships might constrain steel erection. Each interphase concurrence relationship describes the fraction of a phase's total scope that can be done based on the fraction of work released by a supplying phase. Interphase concurrence relationships characterize the dependencies among the off-diagonal terms in the Design Structure Matrix. They are potentially nonlinear, allowing our model to capture changes in the degree of dependence among phases as a project evolves. For example, ASIC designers may be able to develop certain standard elements of the design (memory registers, data bus) with early information about customer requirements, but may be unable to continue until full specifications for the required functionality are released.

Concurrence relationships are characteristic features of a project's network structure and must be estimated for each project. We tested our model against the behavior of a medium-sized chip development project at a major U.S. semiconductor firm (the Python project) [18]. Ford and Sterman describe the protocol used to elicit these concurrence relationships from project personnel and provide examples. Figure 3 shows four expert estimates of the interphase concurrence relationship between the product definition and design phases of the Python project. The product definition phase develops product architecture and specifications based on the Python chip's market and target performance. The designers use these specifications as the basis for the detailed design embodied in the software code used to

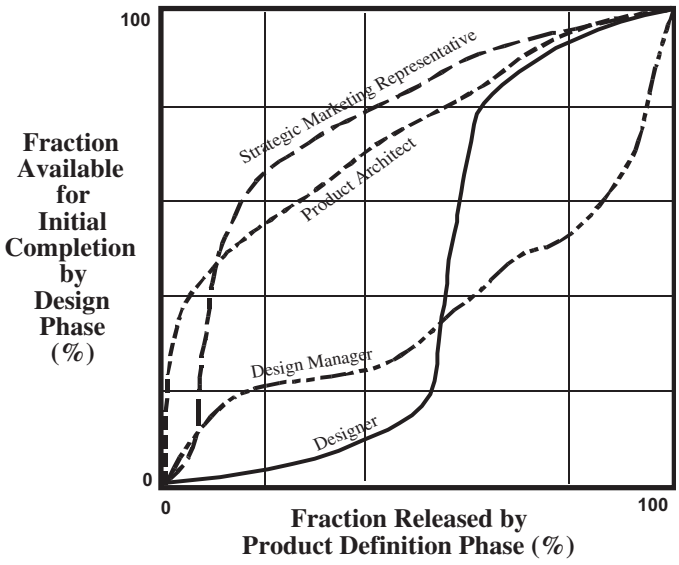


Figure 3. Four estimates of the interphase concurrence relationship between the product definition and design phases.

lay out individual components on the silicon. Each estimate in Figure 3 describes the mental model of a participant concerning the question "How much design work can be completed based on the fraction of the product definition work that has been completed, approved and released?"

Interestingly, the two "upstream" people (the marketing representative and the product architect) believed the "downstream" people (the design manager and designer) could, and presumably should, begin their work quite early, when few product specifications have been released, while those downstream believed their work could only progress when a majority of the specifications were available. These differences in mental models had led to conflict and delay in prior development projects. The explicit description of these mental models initiated and facilitated discussions for improving the organization's development processes.

3.3 Model Testing

The model was tested for structural and behavioral similarity to actual development projects using standard methods [20,37, Ch. 21]. Model structure was based on product development project processes and organizations as described in the literature and derived from our fieldwork. We examined the ability of the model to replicate the experience of the Python project described above. Python applied all the major precepts of concurrent development including overlapping phases and cross-functional teams. The organization was well trained in concurrent development practices [15,40]. Figure 4 compares Python's original schedule and actual performance, developed from records of the phase durations and completion dates (e.g., by counting lines of code in each version of the design code). As is common in semiconductor development, and verified in our interviews, the design phase planned to release its work in a single large package, generating the jump in planned progress 12 weeks after the project began. However after development began the decision was made to design the Python chip in two components instead of one, resulting in two design releases and therefore two jumps in performance. Like the Rattlesnake project, Python suffered from the 90% syndrome. The project 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).

We drew on our fieldwork and prior research (e.g., [11]) to estimate the parameters for the Python case. Ford and Sterman [18] describe the protocol used to elicit the concurrence relationships and provide examples from the Python project. Figure 5 shows planned and actual project performance as simulated by our model. Planned progress simulates management's plan for a single design release, their assumption of no interphase iteration, and overestimation of productivity. Experienced managers expect iteration but are often required to use "stretch objectives" in planning because management believes they keep motivation and pressure to perform high. Python project developers repeatedly described the unrealistic optimism used to plan projects, including the assumption of little or no rework.

The model simulation (Figure 5) closely matches actual project behavior (Figure 4) and recreates the 90% syndrome experienced by the Python project. Differences between our simulation and the project's actual behavior are largely due to resource constraints omitted from the model, specifically staffing problems created by the unanticipated iterations. The similarity between Figures 4 and 5 indicates that even projects staffed by skilled personnel with ample resources can experience the 90% syndrome, solely as a function of the informational and physical dependencies created by concurrency.

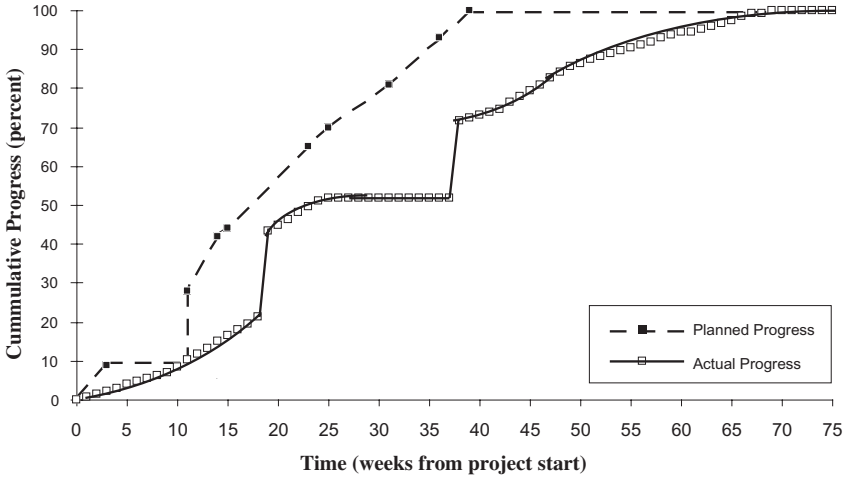


Figure 4. Planned and actual progress of the Python project.

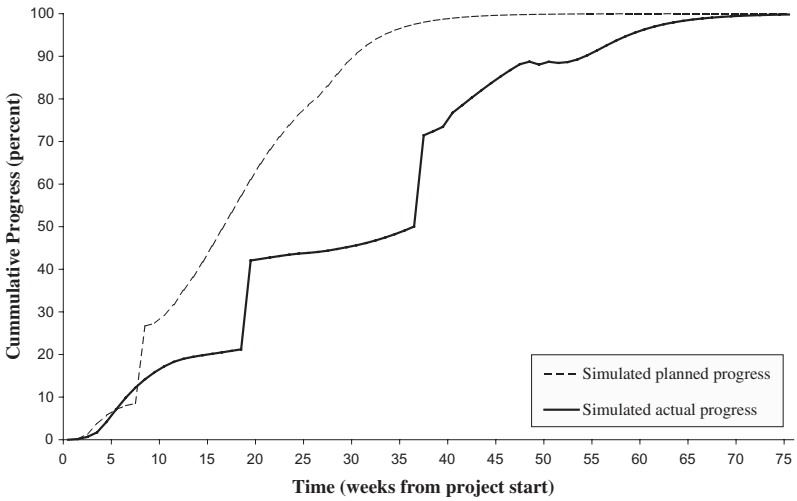


Figure 5. Planned and actual progress of the Python project as simulated by the model.

4. Implementing Concurrent Development Processes

To examine the impact of concurrence on schedule performance, we simulated the model with different levels of concurrence. We vary all inter- and intraphase concurrence constraints. We preserve start time constraints such as cases where a phase cannot start until at least some work has been released by a supplying phase. Figure 6 shows the impact on project duration, iteration, and quality of varying the degree of concurrence from fully sequential (−100% of the base case) to highly overlapped activities (+150% of the base case).[1]

As expected, large reductions in the degree of concurrence cause sharp increases in duration (Figure 6). As overlapping decreases some phases delay the start of their work well after the point at which the supplying phases have completed theirs. Increasing concurrency reduces duration, but with sharply diminishing returns: a 50% increase in overlap compared to the base case reduces duration by 22%, while another 50% increase cuts duration only another 6%, and improvement essentially ceases beyond that point. Figure 6 also shows total work effort relative to total project scope, defined as the cumulative number of tasks completed (both initially and through rework), and is a proxy for project cost.[2]

Note that in the base case, total work effort is 55% greater than project scope due to the impact of rework (if all tasks were completed perfectly with no need for changes work effort would equal project scope). Interestingly, increasing concurrence *decreases* total work effort—a 50% increase in concurrence cuts total work effort from 1.55 to 1.27 times the project scope. One might argue that total work effort should rise with increasing concurrence since more work must be redone when errors are discovered. This effect does occur, but is overwhelmed by another, less intuitive impact: increased concurrence increases the average iteration path length by delaying the discovery of the need for rework to phases farther from the generating phase. In an iteration cycle, rework requirements are passed from the discovering phase to the originating phase. After coordination, changes are made to the flawed work in the originating phase and to contaminated work in all affected phases. The reworked tasks are re-inspected, rereleased and arrive at the location of its discovery again. For example, a test phase may discover an error in the chip and trace it to the design. Test engineers notify and coordinate with the designers to specify the location and characteristics of the flaw. The designers then must rework, recheck and rerelease the design, followed by changes in layout, tape out, masking, and prototype fabrication. The cycle is completed when testing of the redesigned prototype begins. High concurrency means downstream activities carry out a substantial portion of their work before they (or any other phase) have a chance to detect and correct errors. In essence, the downstream phases outrun the discovery of inherited rework requirements.

From the preceding it appears that increasing concurrence both speeds the project and cuts project costs. However, the third graph in Figure 6 shows the cost of concurrency: project quality drops significantly. In the base case, the number of uncorrected errors

[1]A stretch factor of +50% more concurrence means each phase can now do 75% of its work at the point where it could have done 50% in the base case; a stretch factor of −50% means the phase could do only 25% of its work at the point where it could do 50% in the base case. In implementing different levels of concurrence we preserve constraints, where they exist, that, e.g., phase *j* cannot begin its work until phase *i* has released as least some of its work. Such constraints mean the "stretch" factor is nonlinear. The web-appendix provides full documentation.

[2]Project cost is actually the sum of the tasks completed by each phase weighted by the unit cost of tasks in each phase. To preserve confidentiality we report total work–not cost, implicitly assuming the unit cost of tasks in each phase are equal.

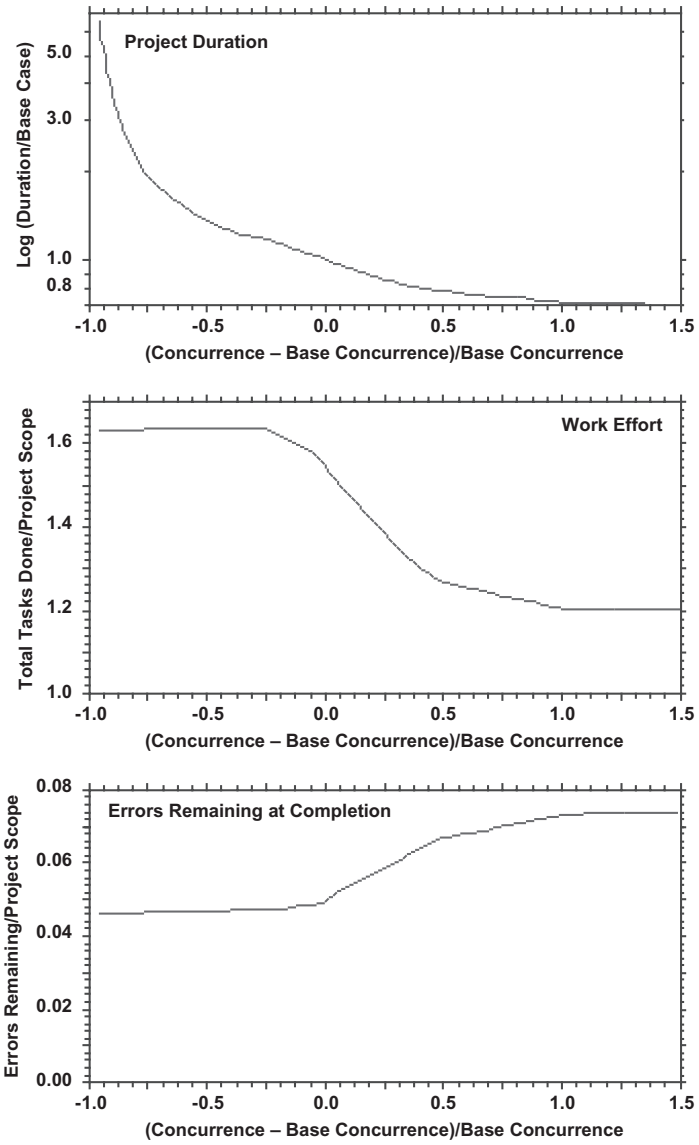


Figure 6. Impact of varying concurrence (internal and external).

released to the customer is 5% of total project scope. These uncorrected errors include both outright defects (where the product does not function as designed) and instances where the design does not correspond to customer requirements. In the chip development context such errors include features the customer wanted that are not available (e.g., power consumption is too high), features that do not function as designed (e.g., a certain combination of inputs gives a fatal error), or design attributes that cause low manufacturing yield. Increasing concurrence 50% raises errors remaining at project completion to 6.7% of project scope—a 34% increase over the base case. At the same time that increasing concurrence delays the discovery and correction of errors, it also increases the likelihood of releasing tasks requiring rework. As concurrence increases, the information, technologies, and components of each phase relies upon as the basis for its work are necessarily less complete, less accurate, and more ambiguous. The number of tasks requiring rework grows while at the same time the ability of personnel in each phase to detect these problems falls, increasing the number of tasks released with errors and thus the chance that needed rework will not be discovered and corrected before the project is completed.

The simulations show a strong tradeoff between schedule and quality performance. Increased concurrency interacts unfavorably with the delays in the discovery of rework needs. The greater the overlap, the more work is completed and released before rework requirements can be detected, leading to more unplanned iteration. Greater concurrence increases the vulnerability of a project to delays in discovering rework and increases the fraction of work requiring such changes. The result is lower suitability to customer requirements and lower product quality.

5. Conclusions

In this paper we use a dynamic model of development projects 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 the critical role of iteration cycles in explaining the 90% syndrome. Our research suggests that an effective strategy addresses the managerial behaviors that cause iteration cycles to constrain progress. Iteration cycles can delay projects by being more in number, longer in the distance which information must travel, slower in traversing that distance, and occurring later than possible. Researchers have proposed process designs to manage iteration cycle number, speed, length, or timing. For example Terwiesch et al. (1998) recommend "a fast process of problem detection, problem solving and engineering change implementation" to increase iteration cycle speed. They suggest "loosening the coupling (dependence) between development activities" and improving the accuracy of preliminary information, both of which reduce the number of cycles. McAllister and Backhouse (1996) suggest redesigning work flows to reduce the number of iteration paths in a project network. However increased concurrence works against all these recommendations.

Our work has several implications for concurrent development research. The model and simulations demonstrate that effective modeling of development processes must include the structure of information dependencies to explain problematic project behaviors. More specifically, the role of iteration cycles in the 90% syndrome demonstrates the need for explicitly including iteration. Future research can identify and test metrics that relate iteration to different forms of project and phase performance and how specific iteration features constrain progress. Most interesting and relevant for managers, the model can be used to study the interaction of the process structure described in this paper and the behavioral decision rules used by engineers and managers under pressure to meet aggressive deadlines (see [19], this issue). In that paper we show how common behaviors such as concealing the need for rework from managers and colleagues interacts with the structure of concurrent development programs to intensify the 90% syndrome, lower product quality, and undercut the benefits of increased concurrency. We argue that sustained improvements in project performance require integration of both the physical and informational structure of concurrent development processes with the behavioral decision rules of the engineers and managers who work within them.

Acknowledgments

The authors thank 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 team for their interest, commitment, and time.

References

1. Abdel-Hamid, T. and Madnick, S. (1991). *Software Project Dynamics, An Integrated Approach*, Prentice-Hall, Inc: Englewood Cliffs, NJ.
2. Abdel-Hamid, T. (1988). Understanding the "90% Syndrome" in Software Project Management: A Simulation-Based Case Study, *The Journal of Systems and Software*, **8**: 319–330.
3. 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.
4. Backhouse, C.J. and Brookes, N.J. (1996). *Concurrent Engineering, What's Working Where*, Gower, Brookfield, VT: The Design Council.
5. Bohn, R. (July–Aug 2000). Stop Fighting Fires, *Harvard Business Review*, **78**(4): 83–91.
6. Browning, T. (Oct. 1999). Sources of Schedule Risk in Complex System Development, *Systems Engineering*, **2**(3): 129–142.
7. Clark, K.B. and Fujimoto, T. (1991). *Product Development Performance, Strategy, Organization, and Management in the World Auto Industry*, Boston, MA: Harvard Business School Press.
8. Clark, K.B. and Fujimoto, T. (1989). Reducing the Time to Market: The Case of the World Auto Industry, *Design Management Journal*, v. **1**(1): 49–57.
9. Componation, P.J., Utley, D.R. and Armacost, R.L. (1999). Prioritizing Components of Concurrent Engineering Programs to Support New Product Development, *Systems Engineering*, Oct 4, 1999, **2**(3): 168–176.
10. Cooper, K.G. (1980). Naval Ship Production: A Claim Settled and a Framework Built, *Interfaces*, **10**(6): 20–36.
11. Cooper, K.G. and Mullen, T.W (1993). Swords and Plowshares: The Rework Cycle of Defense and Commercial Software Development Projects, *American Programmer*, **6**(5).
12. DeMarco, T. (1982). *Controlling Software Projects*, New York: Yourdon.
13. Eppinger, S.D., Whitney, D.E., Smith, R.P. and Gebala, D.A. (1994). A Model-Based Method for Organizing Tasks in Product Development, *Research in Engineering Design*, **6**: 1–13.
14. Ettlie, J.E. (1995). Product-Process Development Integration in Manufacturing, *Management Science*, **41**: 1224–1237.
15. Ford, D.N. (1995). The Dynamics of Project Management: An Investigation of the Impacts of Project Process and Coordination on Performance, Doctoral Thesis, Cambridge, MA: Massachusetts Institute of Technology.
16. Ford, D.N., Hou, A. and Seville, D. (1993). An Exploration of Systems Product Development at Gadget Inc. Report D-4460, System Dynamics Group, Sloan

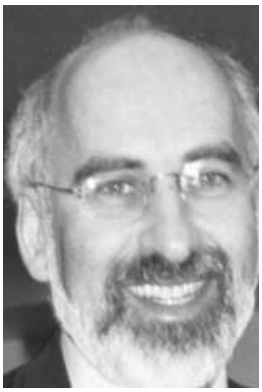
School of Management, Cambridge, MA: Massachusetts Institute of Technology.
17. Ford, D.N. and Sterman, J.D. (1998a). Dynamic Modeling of Product Development Processes, *System Dynamics Review*, **14**(1): 31–68.
18. Ford, D.N. and Sterman, J.D. (1998b). Expert Knowledge Elicitation to Improve Formal and Mental Models, *System Dynamics Review*, **14**(4): 309–340.
19. Ford, D.N. and Sterman, J.D. (2003). The Liar's Club: Concealing Rework in Concurrent Development, *Concurrent Engineering: Research and Applications* (this issue).
20. Forrester, J and Senge, P. (1980). Tests for Building Confidence in System Dynamics Models, *TIMS Studies in the Management Sciences*, **14**: 209–228.
21. Haddad, C.J. (1996). Operationalizing the Concept of Concurrent Engineering: A Case Study from the U.S. Auto Industry, *IEEE Transactions on Engineering Management*, **43**(2): 124–132.
22. Hauptman, O. and Hirji, K.K. (1996). The Influence of Process Concurrency on Project Outcomes in Product Development: An Empirical Study of Cross-Functional Teams, *IEEE Transactions on Engineering Management*, **43**(2): 153–178.
23. 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.
24. Kiewel, B. (January 1998). Measuring Progress in Software Development, *PM Network*, Project Management Institute, **12**(1): 29–32.
25. Krishnan, V. (1996). Managing the Simultaneous Execution of Coupled Phases in Concurrent Product Development, *IEEE Transactions on Engineering Management*, **43**(2): 210–217.
26. Krishnan, V., Eppinger, S.D. and Whitney, D.E. (1995). A Model-Based Framework to Overlap Product Development Activities, *Management Science*, **43**: 437–451.
27. Loch, C.H. and Terwiesch, C. (1998). Communication and Uncertainty in Concurrent Engineering, *Management Science*, **44**(8): 1032–1048.
28. 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.
29. Meyer, C. (1993). *Fast Cycle Time*, *How to Align Purpose, Strategy*, *and Structure for Speed*, New York: The Free Press.
30. Moffat, L.K. (1998). Tools and Teams: Competing Models of Integrated Product Development Project Performance, *Journal of Engineering Technology and Management*, **15**: 55–85.
31. Nevins, J.L. and Whitney, D. (1989). Concurrent Design of Products & Processes, *A Strategy for the Next Generation in Manufacturing*, New York: McGraw-Hill.
32. Noreen, E., Smith, D. and Mackey, J. (1995). *The Theory of Constraints and its Implications for Management Accounting*, Great Barrington, MA: North River Press.
33. Patterson, M.L. (1993). *Accelerating Innovation, Improving the Process of Product Development*, New York: Van Nostrand Reinhold.
34. Rosenthal, S.R. (1992). *Effective Product Design and Development*, Homewood, IL: Business One Irwin.
35. Smith, R.P. and Eppinger, S.D. (1997a). A Predictive Model of Sequential Iteration in Engineering Design, *Management Science*, **43**(8): 1104–1120.
36. Smith, R.P. and Eppinger, S.D. (1997b). Identifying Controlling Features of Engineering Design Iteration, *Management Science*, **43**(3): 276–293.
37. Sterman, J.S. (2000). *Business Dynamics*, *Systems Thinking and Modeling for a Complex World*, New York: Irwin McGraw-Hill.
38. Sterman, J.D. (1994). Learning in and about Complex Systems, *System Dynamics Review*, **10**(2–3): 291–330.
39. Terwiesch, C., Loch, C.H. and De Meyer, A. (2002). Exchanging Preliminary Information in Concurrent Engineering: Alternative Coordination Strategies, *Organization Science*, **13**(4): 402–419.
40. Voyer, J., Gould, J. and Ford, D.N. (1997). Systematic Creation of Organizational Anxiety: An Empirical Study, *Journal of Applied Behavioral Science*, **33**(4): 471–489.
41. Wheelwright, S.C. and Clark, K.B. (1992). *Revolutionizing Product Development*, *Quantum Leaps in Speed*, *Efficiency*, *and Quality*, New York: The Free Press.
42. Womack, J.P., Jones, D. and Roos, D. (1990). *The Machine that Changed the World, The Story of Lean Production*, New York: Rawson Associates.
43. Zirger, B.J. and Hartley, J.L. (1996). The Effect of Acceleration Techniques on Product Development Time, *IEEE Transactions on Engineering Management*, **43**(2): 143–152.

Biographies

David N. Ford

David N. Ford, PhD, P.E. is an Assistant Professor in the Construction Engineering and Management Program in the Department of Civil Engineering, Texas A&M University. He researches development project strategy, processes, and resource management. Dr. Ford earned his PhD from MIT and Master and Bachelors degrees from Tulane University. He has over 14 years of engineering and project management experience.

John D. Sterman



John D. Sterman is the Jay W. Forrester Professor of Management at the MIT Sloan School of Management and Director of MIT's System Dynamics Group. His most recent book is Business Dynamics: Systems Thinking and Modeling for a Complex World.