Appendix: Dynamics of Agile Software Development – Model Structure

This study was conducted within the context of a much broader research effort to study, gain insight into, and make predictions about the dynamics of the entire software development process. A major part of this effort was devoted to the development of a comprehensive system dynamics computer model of software development. The original model is currently being used in several research capacities. Through modeling and simulation, the model is used to study/predict the dynamic behavior of the software development process and of the implications of managerial policies and procedures pertaining to the development of software. In our own research four areas have so far been studied: (1) scheduling; (2) control; (3) quality assurance; and (4) staffing. The exercise produced three kinds of results: uncovered dysfunctional consequences of some currently adopted policies (e.g., in the scheduling area); provided guidelines for managerial policy (e.g., on the allocation of the quality assurance effort); and provided new insights into software project phenomena (e.g., Brooks' Law).

In addition, the model is being used as an "experimentation microworld" to study dynamic decision making behavior in the software management domain, e.g., project planning and control (Abdel-Hamid et al, 1993), and the impact of feedback (Sengupta and Abdel-Hamid, 1993), unreliable information (Sengupta and Abdel-Hamid, 1996), and reward structures (Abdel-Hamid, Sengupta, and Hardebeck, 1994) on performance.

The model was developed on the basis of field interviews of software project managers in five organizations, complemented by an extensive database of empirical findings from the literature. The model integrates the multiple functions of the software development process, including both the management-type functions (e.g., planning, controlling, and staffing) as well as the software production-type activities (e.g., designing, coding, reviewing, and testing).

Figure A shows a high-level view of the model's four subsystems: human-resource management, software production, control, and planning, and some of the relations between them. The actual model is very detailed and contains more than 100 causal links; a full description of the model's structure and its mathematical formulation is published in (Abdel-Hamid and Madnick 1991).

Human-resource management

This subsystem captures the hiring, assimilation, and transfer of people. We segregate the project's work force into employee types (newly hired and experienced, for example). We make this distinction because new team members are usually less productive than veterans. This segregation also lets us capture the training process to assimilate new members. The veterans usually train the newcomers, both technically and socially. This is important, because this training can significantly affect a project's progress by reducing the veteran's productivity. Team members are allowed to quit the project, either because of "normal attrition" or because they are exhausted by working overtime for too long (caused by schedule pressure). The quit rate is defined below.

- Quit rate = (Experienced Workforce / average employment time) * multiplier due to exhaustion
- Average employment time = 700 days
- Multiplier due to exhaustion = GRAPH(Exhaustion,0,5,{1,1,1,1.05,1.10,1.15,1.3,1.5,1.75,2.3,3.8,4.8,6,7.4,9})
- Hiring rate = workforce to replace / hiring delay
- Workforce to replace = initial workforce – (newly hired workforce + experienced workforce)
- Initial workforce = 4.9 employees

- Hiring delay = 40 days

In the model of Abdel-Hamid and Madnick experience is not explicitly modeled. Newly hired and experienced employees have a nominal potential productivity (of respectively 0.5 and 1 tasks/person/day). This potential productivity does not gradually increase when new employees are trained and gain more experience, as we would expect in real-life. Therefore, we included an "Experience" stock in the model. This level of experience is at the start of a new project or part of a project relatively low. Each new team member brings some experience to the team. But because this experience is lower than the currently average experience of the team, bringing in a new team member, decreases the average team experience. Experience decreases when team members quit the team, and it increases each day the team works on a development task. When an iteration is finished, a project part is closed and the team starts with a new part or iteration. At the start of this new iteration, the experience with respect to the new part that needs to be developed is low. Therefore, each time the team starts with a new iteration, the level of experience drops, after which it gradually increases with each day the team develops tasks. It is assumed that this drop in experience is equal to 50% of the experience that was gained during the process of developing the part. The average experience of the team determines the productivity of the team via the learning curve (see also pp. 338 of Sterman, 2000). Experience is measured in working days.

- (d/dt) Experience of Employees = increase of experience by hiring + increase of experience by learning – decrease of experience by attrition – experience decay rate
- Initial experience of employees = initial team size * reference experience of employees
- Reference experience of employees = 1.5 working year

- Increase of experience by hiring = hiring rate * average experience of new hires
- Average experience of new hires = 0.2 working year
- Increase of experience by learning = daily manpower available after training overhead
- Decrease of experience by attrition = average experience of team * quit rate
- Experience decay rate = 0.5 * initial person-days in product (only after a review)
- Average experience of team = Experience of Employees / total workforce

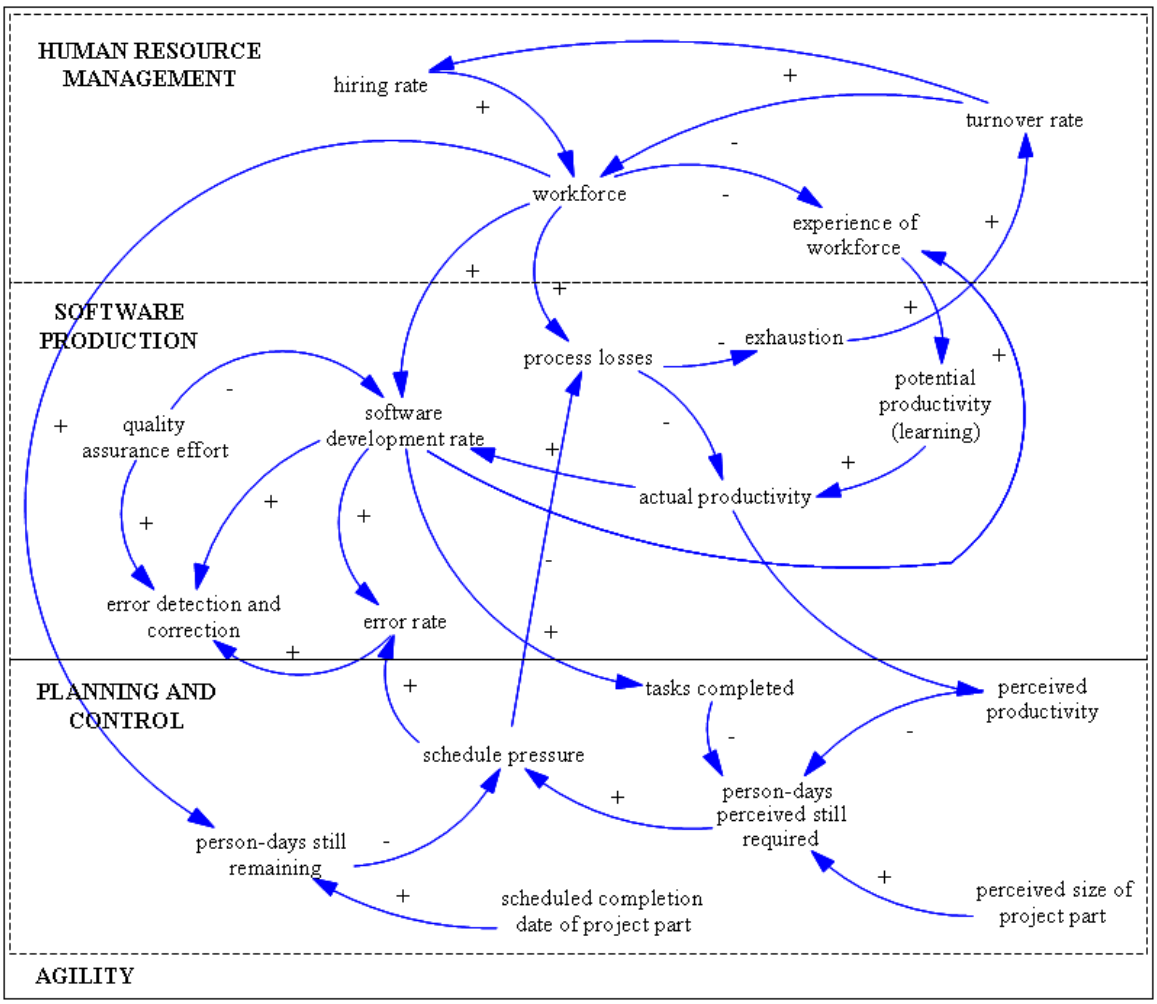

Figure A: Model of Software Project Management

Software Production

This subsystem models development; it does not include the operation and maintenance phases. The development phases included are designing, coding, and testing. As software is

developed, it is reviewed to detect any defects e.g., using quality assurance activities such as structured walkthroughs. Errors detected through such activities are reworked. Not all software defects are detected during development, however; some escape detection until the testing phase. The software production subsystem models productivity and its determinants in great detail. Productivity is defined as potential productivity (due to learning) minus the loss from faulty processes. Potential productivity due to learning is the maximum level of productivity that can occur when an individual or group makes the best possible use of its resources, and is a function of the average level of experience of the team. Losses from faulty processes are losses in productivity from factors such as communication and coordination overhead and low motivation.

- Productivity of employees due to learning = nominal productivity of employees * (average experience of team / reference experience of employees)^exponent learning curve
- Exponent learning curve = LN(1+strength of learning curve)/LN(2)
- Strength of learning curve = 0.25
- Nominal productivity of employees = 0.8 tasks/day/employee
- Software development productivity = productivity of employees due to learning * multiplier due to communication losses and motivation

Control Subsystem

As progress is made, it is reported. A comparison of the degree of project progress to the planned schedule is captured within the control subsystem. In all organizations, decisions are based on the information available to the decision maker. Often, this information is inaccurate. Apparent conditions may be far removed from those actually encountered, depending on information flow, time lag, and distortion. Progress rate is a good example of a

variable that is difficult to assess during the project. Because software is basically an intangible product during most of the development, it is difficult to measure things like programming performance and intermediate work. In the earlier phases of development, progress is typically measured by the rate of resource expenditure rather than accomplishments. But as the project advances towards its final stages, though, work accomplishments become relatively more visible and project members better perceive how productive the work force has actually been.

Planning subsystem

In the Planning Subsystem, project estimates are made and revised as the project progresses. For example, when a project is behind schedule, the plan may be revised to hire more people, extend the schedule, or both. By dividing the value of person-days remaining at any point in the project by the time remaining, a manager can determine the indicated work force level, which is the work force needed to complete the project on time. However, hiring decisions are not made solely on the basis of scheduling requirements. Managers also consider training requirements and the stability of the work force. Thus, before adding new project members, management assesses the project employment time for the new members. In general, the relative weighting between the desire for work force stability and the desire to complete the project on time is not static; it changes throughout the project's life. Although management determines the work-force level needed to complete the project, this level does not necessarily translate into the actual hiring goal. The hiring goal is constrained by the ceiling on new hires. This ceiling represents the highest work force-level management believes can be adequately handled by its experienced project members. Thus, three factors - scheduled completion time, work-force stability, and training requirements - affect the work-force level.

Model Validation

The model was developed on the basis of field interviews of software project managers in five organizations, complemented by an extensive database of empirical findings from the literature. The following sets of tests were conducted to validate the model:

- Face validity test. To test the fit between the rate/level/feedback structure of the model and the essential characteristics of real project environments. This fit was confirmed by the software project managers involved in the study.
- Replication of reference modes. To test whether the model can endogenously reproduce the various reference behavior modes characterizing real environments. Reference modes reproduced by the model included a diverse set of behavior patterns both observed in the organizations studied as well as reported in the literature (e.g., the "90% syndrome", diminishing returns of QA effort, the deadline effect, etc.).
- Case studies. Five case studies were conducted after the model was completely developed[1]. All case studies were conducted in organizations other than the five organizations studied during model development.

References

Abdel-Hamid, Tarek K., and Stuart E. Madnick. 1991. *Software Project Dynamics – an integrated approach*. Prentice Hall, Englewood Cliffs, New Jersey.

Abdel-Hamid, Tarek K., Kishore Sengupta, and Daniel Ronan. 1993. Software Project Control: An Experimental Investigation of Judgment with Fallible Information. *IEEE Transactions on Software Engineering*. 19:6:603-612.

Abdel-Hamid, Tarek K., Kishore Sengupta, and Michael J. Hardebeck. 1994. The Effect of Reward Structures on Allocating Shared Staff Resources Among Interdependent Software

[1] Two case studies are reported in Abdel-Hamid and Madnick (1991) and Abdel-Hamid (1993). Case studies were conducted independently at Bellcore (Glickman and Kopcho 1995), Mitre (Powell 1987), and Bell Laboratories (not published).

Projects: An Experimental Investigation. *IEEE Transactions on Engineering Management*. 41:2:115-125.

Glickman, S. and J. Kopcho. 1995. Bellcore's Experiences Using Abdel-Hamid's Systems Dynamics Model. *COCOMO Conference*. Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University.

Powell, F.D. 1987. Study of a Software Development Process Dynamic Model. Technical Report MTR 10314, Mitre Corporation, December.

Sengupta, Kishore, and Tarek K. Abdel-Hamid. 1993. Alternative conceptions of Feedback in Dynamic Decision Environments: An Experimental Investigation. *Management Science*. 39:411-428.

Sengupta, Kishore, and Tarek K. Abdel-Hamid. 1996. The Impact of Unreliable Information on the Management of Software Projects: A Dynamic Decision Perspective. *IEEE Transactions on Systems, Man, and Cybernetics*. 26:177-189.

Sterman, John D. 2000. *Business Dynamics - Systems Thinking and Modeling for a Complex World*, Boston: Irwin McGraw-Hill.