MODELLING THE IMPACT OF QUALITY INITIATIVES OVER THE SOFTWARE PRODUCT LIFE CYCLE

Thomas Fiddaman and Rogelio Oliva

Sloan School of Management
Massachusetts Institute of Technology
One Amherst St., Room E40-294
Cambridge, MA 02139, USA

R. Rembert Aranda

Aragon Associates, Inc.
24 Redfield Circle, Box 449
Derry, NH 03038, USA

ABSTRACT

This paper describes a System Dynamics model which forms the basis for a management flight simulator that explores the impact of two total quality initiatives, Formal Inspection and Quality Function Deployment, on the software development and adoption process. The paper focuses on the new perspective on software development dynamics gained in the construction of the model. It describes the measures of performance used and the causal structure of selected sectors. The model links existing software project management and market diffusion structures, adding an explicit representation of product functionality and evolving customer requirements based on Kano's Dimensions of Quality diagram. A discussion of future goals for this research and an evaluation of the impact of this kind of work on the software industry is presented.

MODEL FOCUS AND OBJECTIVES

In general, current process and quality improvement efforts in the software field have a time horizon and substantive focus that includes only a single development project. As a result, software developers operationally regard processes that have long time scales or that cross organizational boundaries as beyond the scope of their concern. This limited perspective inhibits the capacity of a software development group to learn about long-term processes and to understand its relationships with other functional groups in its organization and with customers.

This bias is reflected in the kinds of models now available to software project managers. The planning models most widely used today are statistical models that forecast schedule and resource requirements for one software release. They exclude consideration of customer satisfaction measures and rely on curve-fit historical data for prediction (Boehm, 1981; Putnam, 1989). Such models do not address an important quality variable, the gap between delivered functionality and user requirements at the time of software release. Nor do they consider the evolutionary nature of customer requirements and the dynamics of software markets.

We are currently engaged in building a Management Flight Simulator (Diehl, 1992b; Graham, Morecroft, Senge, & Sterman, 1992) to support policy and management decisions about the use of Total Quality Management techniques for software development. The Flight Simulator will be used as a "Software Product Quality Microworld", a computer-based learning environment enabling cross-functional product development teams to practice decision making in accelerated time. The simulator extends the traditional view of the software development process to include multiple releases of a product, market diffusion dynamics, and the evolution of customer requirements.

The simulator will provide a common vocabulary and set of experiences upon which to base discussion and encourage consideration of the short-term costs of quality initiatives in the context of their long-term dynamic benefits. In using such a simulator, a cross-functional team also has the opportunity to rehearse how to cooperate better, deepening its understanding of how local decisions affect other functions and product success in the long term. The simulator is appropriate for training rather than for operational decision support, because the underlying model currently describes general rather than precise behavior (Graham, et al., 1992). While it contains hypotheses and assumptions which are still tentative, we have found it to be useful to software developers implementing quality programs.

MODEL DEVELOPMENT

Our model builds on earlier system dynamics models by Tarek Abdel-Hamid on software project dynamics (1991) and by Winston Ledet on the software product life cycle at Digital Equipment Corporation (Aranda, 1991; Aranda & Ledet, 1992; Ledet, 1992). Tarek Abdel-Hamid's model provides an operational representation of a single-release development project, with explicit representation of important processes such as the rework of errors and delays in hiring and training which are absent in traditional models. Winston Ledet's model adds multiple product releases, the evolution of customer requirements, and market diffusion dynamics (Bass, 1969; Sterman, 1991). We completely revised the model formulations and extended the concept of requirements evolution. The model was initially developed as part of our work at Digital Equipment Corporation's Software Engineering Group.

The model was developed using the iTHINK™ (Richmond, Peterson, & Charyk, 1992) system dynamics modelling software, and the Microworlds™ Creator and S**4 products (Diehl, 1992a; MicroWorlds, 1992) for modelling and simulator interface development. The model has approximately 750 relationships, divided into 20 sectors, with 75 stocks. While the structure of the model is conceptually clear, constructing the model involved many difficult formulation issues. This was principally due to the difficulty of mixing continuous processes, like product adoption and program code generation, with discrete processes, like the release of a new version of a product.

We used Soft Systems Methodology (SSM) to help frame the scope and objectives of the dynamic modelling work. SSM (Checkland, 1988; Wilson, 1984) was developed at the University of Lancaster in the 1970's as engineers found it necessary to extend the system approach to tackle organizational problems. SSM is a methodology to elicit, evolve and shift mental models. SSM proved effective in conjunction with system dynamics modelling, especially in drawing the bounds of the relevant systems and focusing on key behaviors.

TOTAL QUALITY MANAGEMENT IN SOFTWARE DEVELOPMENT

Over the past five years, Total Quality Management (TQM) techniques have been adopted by many software development organizations. TQM techniques arose in relatively mature product settings where requirements are more stable and production processes are more repetitive and longer lived than is typical in software development. The Quality Movement has spawned three types of techniques: Quality Control (Fitness to Standard), Reactive Improvement (Continuous Improvement), and Proactive Improvement (Fitness for Latent Requirements) (Shiba, Graham, & Walden, forthcoming 1993).

Most of these quality approaches emphasize greater rigor of the software engineering process. A more recent set of initiatives places emphasis on listening to the "voice of the customer" through Quality Function Deployment (QFD), a structured way to discover software product attributes desired by customers and to map these to engineering functional specifications (Hauser & Clausing, 1988; Zultner, 1992). Aligned with the trend towards TQM is the adoption of performance metrics that address a richer set of business issues than traditional measures of program error density or project schedule fulfillment.

Interest in TQM and process improvement is spreading in software circles, but the goals for such programs are too often set by the benchmark of the earliest implementors. Theoretical frameworks are needed to evaluate the strategic impacts of these policies in such a dynamic and constantly-evolving working environment as software development. System

Dynamics modelling is a powerful medium for exploring and testing the trade-offs and potential benefits of using a particular strategy at different stages of a software product's life.

In order to effectively manage TQM initiatives and the trend towards integrated software metrics, we need a shared view and common language among four culturally and administratively separate groups: development engineers, quality assurance personnel, business and finance managers, and customer contact personnel (support, sales, and marketing). The contribution of each group to product development decisions varies over the product life cycle. For example, customer-focused reliance on users to drive product design may produce excellent results in a mature product environment, but disappointing consequences when launching an innovative product that customers have little experience using. Similar problems can arise from allowing engineering innovation or time-to-market considerations to predominate in product planning decisions.

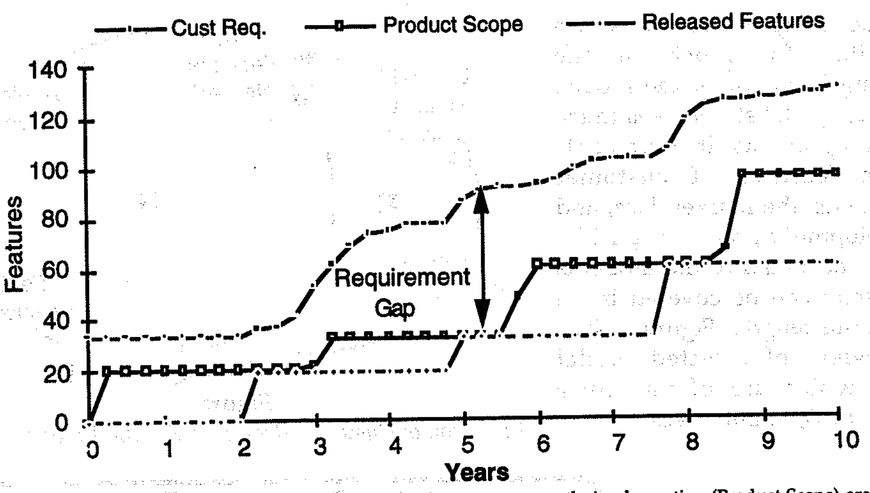
MEASURING SUCCESS

Two questions dominate current debate about software management: how to complete projects on schedule and within budget. This focus reflects that today's software process and quality improvement efforts generally treat the boundary of software development systems as beginning with accepting a set of requirements and ending at the conclusion of a project to meet those requirements (Davis, Bersoff, & Corner, 1988; Nguyen, Clough, Ahmed, Smith, & Vidale, 1992, are conspicuous exceptions). This view, borrowed from engineering disciplines, implies that the rate at which requirements change is negligible – or at least slower than the rate at which software to meet the requirements can be developed. This view also implies that engineering success ought to be judged by success in meeting the engineering requirements formally stated in a functional specification at project start rather than the user requirements at time of project conclusion.

This prevailing boundary definition is overly restrictive and omits some of the most critical quality and productivity improvement questions facing the software development community. Therefore, we have focused the model development to address broader strategic questions. This is reflected in our choice of two principal outcome measures for evaluating the simulated outcomes of policy experiments: the fit of product features to customer requirements, and the Return Map.

Fit of Product Features to Customer Requirements

Software developers aim to create product features that meet customer needs, but customer requirements are a moving target. This can have profound strategic consequences for software companies. A company which allows the gap between customer requirements and their product's features to grow large becomes vulnerable to competitive product introductions. The rate of change of customer requirements is an important consideration when weighing additional functionality of a new release against the time required to implement it. Figure 1 shows the varying gap caused by growth in customer requirements and product features as generated by the model. The time scale is ten years, typically representing several releases from the first in a product category (e.g. spreadsheets) to the last supported release.



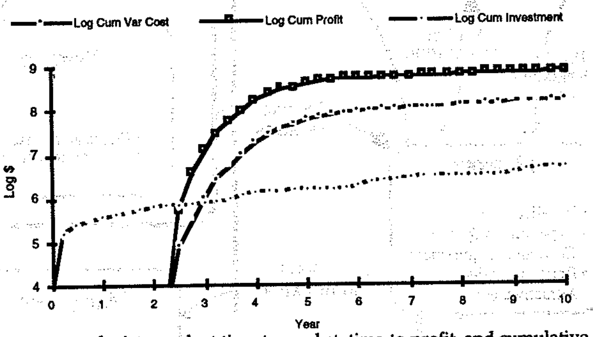
Customer Requirements and the set of features developers are currently implementing (Product Scope) are constant until a product is released (around the end of the second year in this simulation run). The developer sets a new Product Scope target for the next release by the end of the third year, and begins design and testing. In the meantime, customer requirements are increasing. In the last quarter of the fourth year, the developer again releases a new version of the product (cf. Davis, 1988).

Figure 1

Evolution of Customer Requirements and Product Features

The Return Map: Time-to-Profit and Life Cycle Profits

In order to study the long term financial implications of different policies, we adopted the 'Return Map' (House & Price, 1991) as a principal indicator of performance. The Return Map helps key internal groups focus on the inevitable reality that product success as a whole depends on their cooperation in creating value for enterprise customers and shareholders. The Return Map is most useful in conjunction with the product adoption graph, which provides a context in which to compare absolute revenue against unit share of adoption and relative standing in the market. Figure 2 shows a software product Return Map generated by the model. Revenues are a product of unit sales and price, and so are a measure of purchasers who regard the product as offering competitive value.



The return map depicts product time-to-market, time-to-profit, and cumulative profit. Much of the development investment occurs during creation of the first release. When the product reaches the market, revenues begin to accumulate, and cumulative profit becomes positive shortly thereafter. Variable costs dominate investment later in the product life cycle.

Figure 2

The Return Map

OVERVIEW OF MODEL CAUSAL STRUCTURE AND BEHAVIOR

Our model can be summarized in terms of two feedback processes, shown in Figure 3. A positive loop drives the growth of customer requirements and released functionality. The software development process is a negative feedback loop which closes the gap between customer requirements and delivered product features.

These simple processes are responsible for much of the interesting behavior of the model. The actual model structure is much more complex, as it represents multiple features of customer requirements, the marketplace, and the development process explicitly. The size and detail of the model is greater than can be covered in an article of this length. Figure 4 gives an overview of selected model features which are of particular relevance to TQM initiatives.

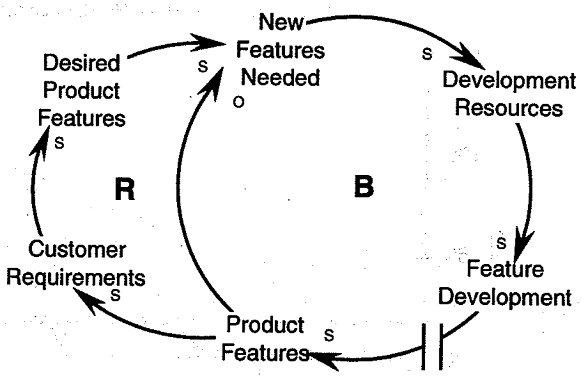


Figure 3
Evolution of Customer Requirements and Product Features

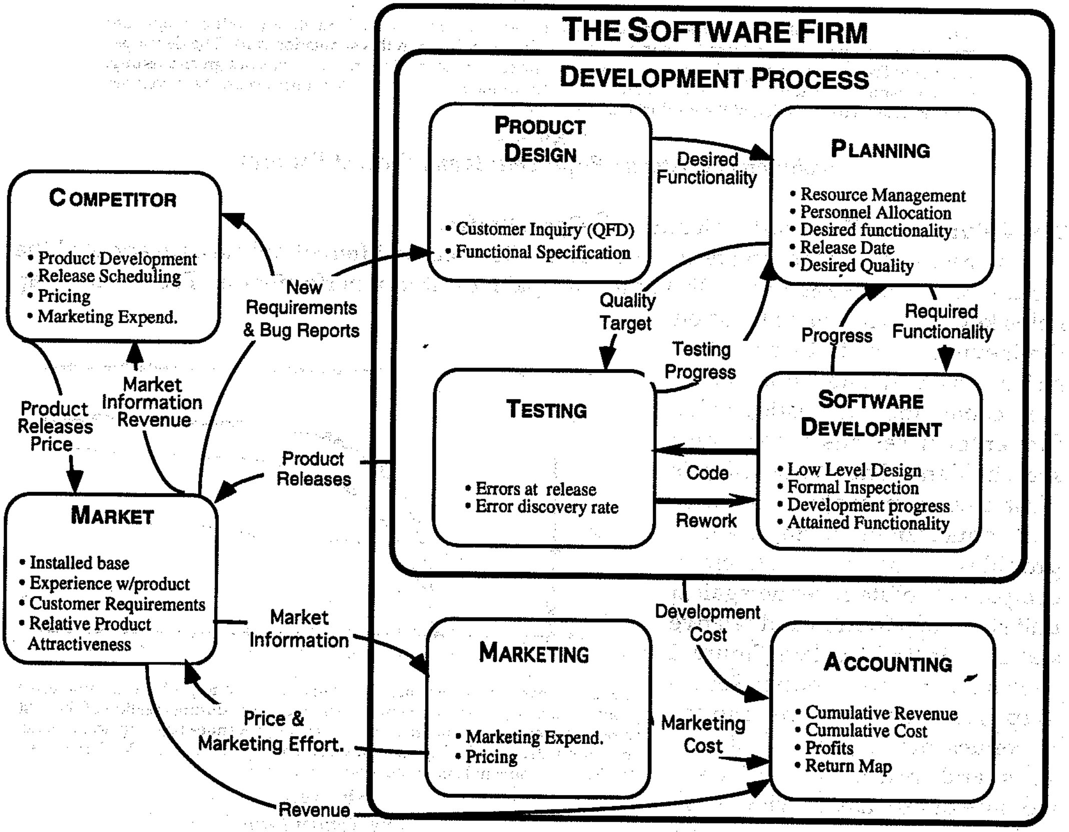


Figure 4
Subsystem Diagram

Customer Requirements Evolution

Our model incorporates a simple dynamic theory of requirements evolution which is graphically summarized in Figure 5. Customer requirements are disaggregated into three types. Two of these, Must-Be's and Delighters, are taken from the Kano (1982) diagram. The third, Noise, represents features which are not widely desired by customers. Delighters

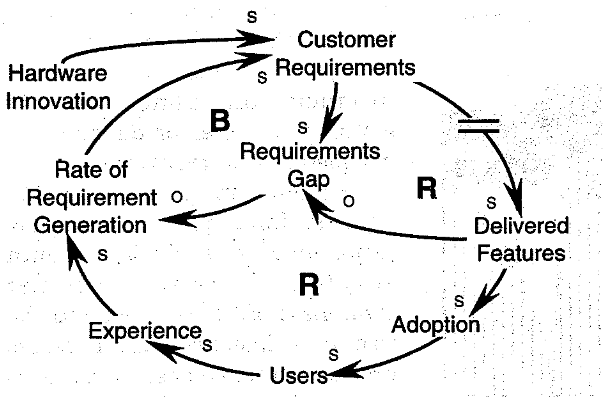


Figure 5

Evolution of Customer Requirements.

As customers gain experience with existing products, they generate new requirements. The set of new requirements that can be generated is bounded by the existing feature set, so the rate of new requirements generation diminishes as requirements exceed delivered features. After a delay, new product features are delivered, creating the potential for further requirements evolution.

increase the attractiveness of a product, while the absence of Must-Be features or an excessive ratio of Noise features to total features decrease attractiveness.

Delighters are generated by customer experience with a product. They are also generated through software developers' innovation. Delighters evolve to become Must-Be's as customers come to expect their presence. Must-Be's are also exogenously generated by innovations on the hardware platform that generate new requirements for compatibility. Noise features arise when customers generate requirements which are not widely needed and when developers' innovations do not match customer requirements.

Software developers seek to match the attributes of their product to customer requirements. They gain knowledge of customer requirements through user complaints and customer inquiry. Known customer requirements and developer innovations are incorporated into the product scope (the set of features to be included in the next release) through the functional design process. Once the product scope has been set, software developers generate a low-level design and program code that implement the desired features. When the product is released, the portion of features in the product scope that have been completed become part of the stock of released features. The behavior resulting from the evolution of customer requirements and product development is shown in Figure 1.

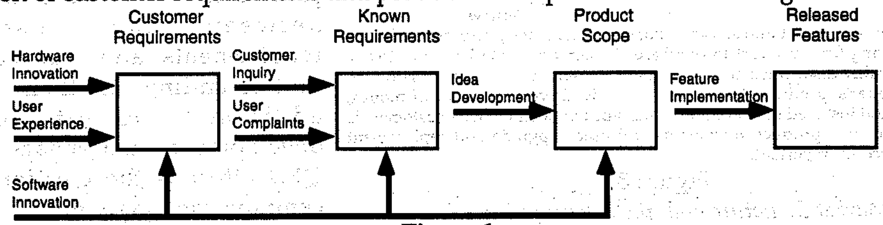


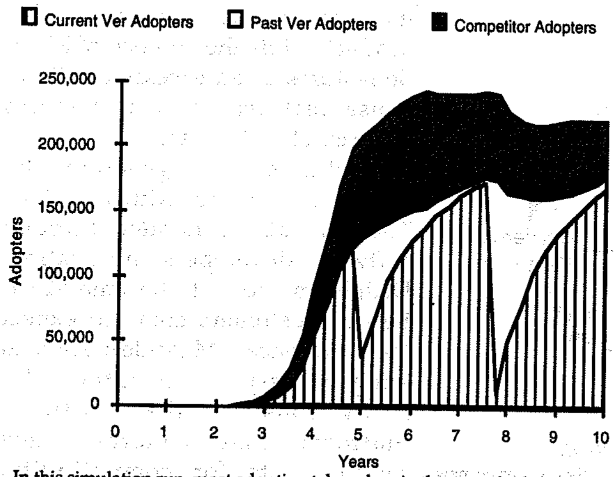
Figure 6

Requirements Evolution, Knowledge Acquisition, and Product Development

Requirement change is fueled by an increasing number of first-time users who find and develop new uses for the product and evolve a set of desired product attributes. The rate of change slows as the maturity of the installed base increases and a smaller proportion of users are innovators. These dynamics can be anecdotally illustrated by thinking back to the first time you used a new type of software product and considering how differently you would have described desired product attributes then than you would have after several months of intensive use. If over the same time period we'd asked for your requirements for an automobile (a mature product) they would have changed comparatively little.

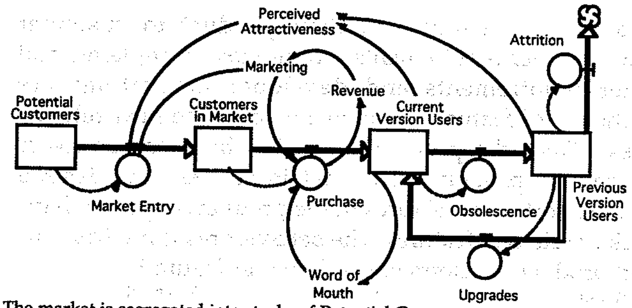
Product Adoption

In the market sectors, the principal developer and a competitor set prices and market their products. The released products are evaluated and adopted by consumers according to a



In this simulation run, most adoption takes place in the second release, at the end of the third year. When the second version is released, current version users of our product become previous version users. The profile of the graph is the Total Adopters in the Market.

Figure 7
Product Adoption



The market is segregated into stocks of Potential Customers, customers who are actively looking for a product to purchase (Customers in Market), and Users. Users are further disaggregated into users of current and previous versions of each developer's product. Growth in sales is driven by word-of-mouth, marketing, and installed base effects. As the stock of potential new customers is depleted and new product versions are released, upgrade and replacement purchases become important.

Figure 8
Market Structure and the Adoption Process

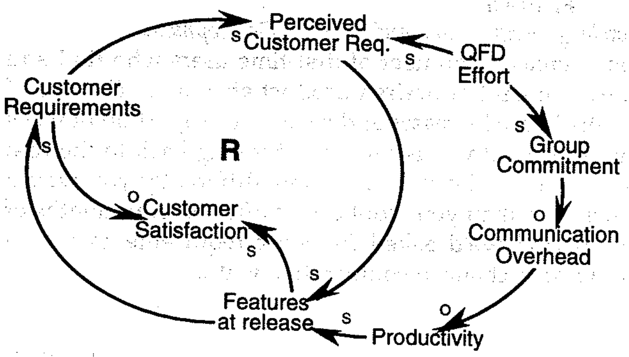


Figure 9
Quality Function Deployment

market diffusion model (Bass, 1969; Mahajan, Muller, & Bass, 1990). Purchasers compare products on dimensions of features, price, error density and installed base attractiveness.

Figure 7 shows the adoption process. Initially, unit sales grow exponentially as word-of-mouth effects multiply consumer awareness of the product and the size of the installed base increases product attractiveness. Purchases then decline as market saturation is reached, moderated by replacement purchases (upgrades and product switching).

Impact of QFD and FI initiatives

Quality Function Deployment

Originally developed in Japan, QFD is a technique for allowing the "voice of the customer" to drive the design process; reports of its success are extensive (Zultner, 1992). QFD is a proactive effort to interview and observe customers to discover their real requirements. The basic assumption is that this process will help 'close the gap' between the customer requirements and our current understanding of them. An additional benefit for a development team engaged in a QFD effort is the creation of a common language and a shared understanding of product direction. A team that goes through this process works more effectively, at least in the design stages of a particular release. Figure 9 illustrates these assumptions as a causal loop diagram.

The QFD process also has costs. It takes time for the development team to go out and interview the customers, (reducing development resources and hence increasing development time and cost), and substantial rework and new work may be generated for developers. An inherent limit to QFD benefit is the requirement change rate, i.e. how much requirements have evolved since the last QFD, which depends upon recent releases, time since release, and adopter experience.

Formal Inspection

Formal Inspection (FI) (Cohen, 1991; Fagan, 1986) focuses on two key processes: the transformation of customer requirements in the product specification to functional units in the design, and the transformation of design elements to working code modules. This has been modeled as a reduction in the design and coding error generation rates based on the fraction of coding and design team resources allocated to FI. The benefits of FI include greatly reduced need for product testing and error correction, lower development and support costs (errors are less expensive when they are corrected earlier), and higher quality as perceived by customers. This effect is retained in base code reused in subsequent releases. Like QFD, FI has costs and diminishing returns. FI requires development resources and hence increases development time and cost in the short run.

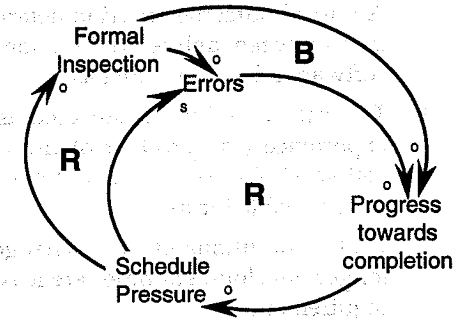


Figure 10
Formal Inspection

CONCLUSIONS & NEXT STEPS

We believe that system dynamics is a promising avenue for the software development community to better understand how the change of requirements is coupled to development policies, and to conduct experiments aimed at incorporating this knowledge into functionality release and quality policies. Many of the benefits of quality and process-improvement initiatives materialize over a longer time frame than a single project, and we believe that system dynamics models are a helpful tool for software developers to understand and balance their long-term benefits and short-term costs. We regard our model as a step toward realizing the potential gains of a systems perspective.

The current software development focus on process (the engineering view) to the exclusion of outcome (end-user and enterprise shareholders' views) leaves unchallenged core assumptions of process improvers about the outcome of greater rigor and process maturity. The broader focus of our model reveals the possibility that in situations where requirements are highly volatile, greater process rigor may result in producing software that is less satisfactory to users—and hence less commercially viable. The tradeoffs among quality, functionality, and development resources, critical in establishing value-added in end user's eyes, is better understood with a view that includes the variable stability of requirements.

At some stages of the product life cycle end-users may value quick delivery of limited-functionality code with a relatively high error density. At other stages end users value later delivery of higher-functionality and lower-error density software. The postulated requirements evolution dynamics provide a framework for discovering the level of process rigor (e.g. FI and process maturity standards) and customer interaction (e.g. QFD) suited to the conditions in a particular development effort. S~me tentative guidelines that have evolved from this work are:

- It is useful to consider a product life time frame (from initial release to retirement) rather than a project (single release) time frame when evaluating process and quality improvement programs

- A critical outcome criterion determining customer satisfaction is the size of the gap between delivered functionality and user requirements at the time of software release (or purchase)
- The rate at which requirements change is a function of cumulative user experience (the product of the number of users and their time spent using software). As users learn and gain experience, latent requirements are revealed in observable forms
- Under conditions of rapid change of user requirements, strategies that cause longer development times are least likely to result in software that satisfies user requirements
- QFD and other customer-driven requirements gathering methods are limited by the level of expressible or observable requirements at any point in time
- FI and other rigor-oriented process enhancements will produce greatest return under conditions where the rate of requirements change enables longer code life (including reuse strategies)
- High-rigor strategies favor the ability to more quickly deliver a given level of functionality in a later release, at the price of later delivery of the initial release

At this early stage we have identified many opportunities for research and for improvements in the model. These include testing the usefulness of the flight simulator and the "feel" of model behavior with a group large enough to reflect the diversity of views among software developers, and validating the hypothesized requirements evolution dynamics against a range of historical software products. We anticipate that the latter effort will benefit from experimentation with alternative formulations of product functionality, particularly with regard to distinguishing the point at which a product enhancement is usefully seen as the beginning of a new class of product rather than as a later version of an existing product. It would also be of interest to investigate the utility and extensibility of our model's concepts for describing the evolution of requirements of other product types.

ACKNOWLEDGMENTS

The authors would like to thank John Sterman for guidance on the flight simulator project discussed in this article, and also on earlier modelling upon which the model draws, especially that carried out by Winston Ledet. We are also grateful to Tarek Abdel-Hamid for review comments at various stages of the work, and his assistance in incorporating structures from his seminal model as the basis for the development sectors of our model. We had the benefit of Ernst Diehl's assistance and review comments regarding Microworlds' development and plans. We also appreciate the review and comments of Nghia Nguyen, Anne Clough, Shuur Ahmed, Brad Smith and Richard Vidale. Finally, thanks are due to numerous individuals in Digital Equipment Corporation for sharing useful perspectives and data.

REFERENCES

Abdel-Hamid, T. K. and S. E. Madnick. 1991. Software Project Management Dynamics: An Integrated Approach . Englewood Cliffs, NJ: Prentice Hall.

Aranda, R. R. (1991) *Overview of TNSG Systems Thinking Project: A Customer-Centered Model of the Software Business.* internal document, Nashua, NH:Digital Equipment Corp. Software Engineering Group.

Aranda, R. R. and W. J. Ledet (1992) *Systems Modelling & TNSG Microworlds.* presentation at Digital Equipment. Corporation Concurrent Engineering Lecture Series. March 19, 1992.

Bass, F. M. 1969. A New Product Growth Model for Consumer Durables. *Management Science,* Vol. 15 : 215-227.

Boehm, B. W. 1981. *Software Engineering Economics* . Englewood Cliffs, N.J.: Prentice-Hall.

Checkland, P. B. 1988. Soft Systems Methodology: An Overview. *Journal of Applied Systems Analysis*, Vol. 15 : 27-30.

Cohen, L. 1991. *Inspection Moderator's Handbook* . Digital Equipment Corporation.

Davis, A. M., E. H. Bersoff and E. R. Corner. 1988. A Strategy for Comparing Alternative Software Development Life Cycle Models. *IEEE Transactions on Software Engineering*, Vol. 14 **(10)**: 1453-1461.

Diehl, E. W. 1992a. *MicroWorld Creator™ 2.0 User's Guide and Tutorial* . Cambridge, MA: MicroWorlds, Inc.

Diehl, E. W. 1992b. Participatory Simulations software for managers: The design philosophy behind MicroWorld Creator. *European Journal of Operational Research*, Vol. 59 **(1)**: 210-215.

Fagan, M. E. 1986. Advances in Software Inspections. *IEEE Transactions on Software Engineering*, Vol. 12 **(7)**: 744-751.

Forrester, J. W. 1961. *Industrial Dynamics* . Cambridge, MA: Productivity Press.

Graham, A. K., J. D. Morecroft, P. M. Senge and J. D. Sterman. 1992. Model supported Case Studies for Management Education. *European Journal of Operational Research*, Vol. 59 **(1)**: 151-166.

Hauser, J. R. and D. P. Clausing. 1988. The House of Quality. *Harvard Business Review*, Vol. 66 **(3)**: 63-73.

House, C. H. and R. L. Price. 1991. The Return Map: Tracking Product Teams. *Harvard Business Review*, Vol. 69 **(1)**: 63-73.

Kano, N., N. Seraku and F. Takahashi (1982). Attractive Quality and Must-Be Quality. In *Nippon QC Gakka 12th Annual Meet (1982)*.

Ledet, W. J. (1992) *Balancing Time-To-Market, Customer Understanding, and Creative Innovation*. Master's Thesis, Sloan School of Management, MIT.

Mahajan, V., E. Muller and F. M. Bass. 1990. New Product Diffusion Models in Marketing: A Review and Research Directions. *Journal of Marketing*, Vol. 54 **(Jan)**: .

MicroWorlds, I. 1992. *MicroWorlds S**4™ User's Gude and Tutorial* . Cambridge, MA: MicroWorlds Inc.

Nguyen, N., A. Clough, S. Ahmed, B. Smith et al (1992). The Software Game: A Dynamic Process Model. In *Systems Thinking in Action Conference*, . Cambridge, MA: Pegasus Communications.

Putnam, L. H. 1989. Strategic Issues in Managing Software Cost and Quality. *Engineering Management Journal*, Vol. 1 **(4)**: 9-18.

Richardson, G. P. 1991. *Feedback Thought in Social Science and Systems Theory* . Philadelphia: University of Pennsilvania Press.

Richmond, B., S. Peterson and C. Charyk. 1992. *iTHINK User's Guide* . Hanover, NH: High Performance systems.

Rubin, C. W. 1992. Quality: The competitive Advantage? *American Progammer*, Vol. 5 **(2)**: .

Shiba, S., A. Graham and D. Walden. forthcoming 1993. *A New American TQM: Four Practical Revolutions in Management* . Cambridge, MA: Productivity Press.

Sterman, J. D. 1991. *B&B Enterprises Management Flight Simulator: An Interactive Simulation for New Product Management* . Cambridge, MA: Sloan School of Management, MIT.

Wilson, B. 1984. *Systems: Concepts, Methodologies and Applications* . Chichester, UK: John Wiley & Sons.

Zultner, R. 1992. QFD for Software: Satisfying Customers. *American Programmer*, Vol. 5 **(2)**: .