System Dynamics Applied to the Modelling of Software Projects.

Article *in* Software - Concepts & Tools · January 1994

Source: DBLP

CITATIONS
22

READS
223

2 authors, including:

Dietmar Pfahl
University of Tartu
191 PUBLICATIONS **2,786** CITATIONS

SEE PROFILE

SYSTEM DYNAMICS APPLIED TO THE MODELLING OF SOFTWARE PROJECTS

Hélène Waeselynck*
INRETS-CRESTA
20, rue Elisée-Reclus
59650 Villeneuve d'Ascq, France
Electronic Mail: waeselyn@terre.inrets.fr

Dietmar Pfahl
Siemens AG, Corporate Research and Development,
System Technologies, Application Center Software,
81730 Munich, Germany
Electronic Mail: Dietmar.Pfahl@zfe.siemens.de

* Guest researcher at Siemens AG at the time of the study reported here.

Abstract

System dynamics is an approach to the modelling and simulation of non-physical systems. In the past, typical application domains were social, biological, or economical systems. Recently, a new application domain has emerged: software development. The report summarises the major results of a study performed to review world-wide activities in the field of system dynamics modelling and simulation applied to software development projects. The basic concepts of the system dynamics approach are presented. Five potential uses of the method for software development projects are identified, discussed and judged with respect to their relevance. The most promising application areas for industrial purposes are training and management support for strategic decisions.

Keywords: Feedback Structures, Modelling, Software Project Management, Simulation, Software Development Process, System Dynamics, Systems Thinking.

I Introduction

The so-called "software crisis" denotes recurrent problems of software cost overruns, late delivery and users' dissatisfaction: the improvement of productivity and quality remains a big challenge in the software industry.

Two factors may affect software project performance. The first one relates to **technological issues**: languages and tools, code reuse, elaboration of standards, etc., and significant progress has already been achieved in this field. However, it has become apparent that the benefit brought by technological advances had its limit, and a second direction for investigation has emerged, namely the improvement of **managerial techniques**. A major characteristic of management is decision making: choosing the right technology, planning and supplying resources, training people, and finding the appropriate strategy to introduce innovative techniques and methods. Unfortunately, the introduction of new development tools, techniques, methods, and processes in real projects is risky; testing their implications requires a tremendous investment in both time and money, without any guarantee that the result is worth the price. This difficulty arises for two reasons. Firstly, large modern software systems, like operating systems or telecommunication systems, are characterised by a huge inherent dynamic

complexity which is hardly understood in all its implications by any single person. Secondly, the underlying process for the development of such systems is carried out by teams of developers, i. e. human beings who have to communicate and cooperate with each other and with their customers in order to meet their project goal. In other words, the development process can be interpreted as the abstraction of a very dynamic and complex socio-technical system which is controlled and guided by the management task.

Now, how do managers identify the most effective and efficient policies to apply on their socio-technical system? Generally, in large-scale industrial software production environments it is much too costly, if not impossible, to experiment with alternatives on real projects. When experimentation on physical systems happens to be unfeasible, a common engineering practice consists of building a model that can be studied by simulation. The model is a mathematical abstraction that acts as a substitute for the real system but is more amenable to manipulation. It is tempting to **adopt the principles of modelling and simulation** to study the general process by which software projects are managed. But here the target system is no longer purely physical: management is a human activity that interacts with other human activities. So, does it mean that simulation is out of scope?

During the 1950s, a modelling method, **system dynamics**, was developed to tackle socio-economic problems. It is based on the assumption of the ubiquity of feedback processes in human interactions: considered from a high level of abstraction, a socio-economic system can be modelled as a feedback structure, whose complex behaviour is generated by the interaction of many nonlinear loops over time. System dynamics aims at identifying organisational (structural) problems, and at exploring through simulation corrective policies that improve the system behaviour. The method has been applied to a wide range of domains, from the management of production-distribution systems to the management of ecosystems. **Recent work has focused on the modelling of software development projects.**

Since the purpose of system dynamics is to analyse the behaviour of complex nonlinear feedback systems, and to look for efficient policies that improve supplied behaviours, intensive computer simulations of mathematical models have to be performed. We were interested in gaining closer insight into the method from an industrial point of view, in order to identify the potentialities offered by system dynamics models in the field of software project management. Of course, when talking about industrial software projects we assume a certain level of maturity, e. g. maturity level two or higher according to the SEI capability maturity model [SEI 91], since we are aware that any modelling attempt based on an ad-hoc development process would be likely to fail.

This report synthesises the result of our investigation after intensive review of the literature and examination of several tools that support the method. The text is organised as follows. In the next section we introduce the underlying concepts of system dynamics models. The mathematical background, problems of modelling and validation, a list of tools, and the major application domains of the system dynamics method are outlined in Section III. Section IV presents the existing system dynamics models of software projects, and Section V discusses the potentialities offered by such models for managerial issues in the software industry.

II Concepts underlying system dynamics

"System Dynamics is the application of feedback control systems principles and techniques to managerial, organizational, and socio-economic problems" [Roberts 78b]. Pioneered at MIT by Jay Forrester during the 1950s [Forrester 61], this approach originated from the idea that feedback mechanisms also exist in business organisations. It took advantage of the engineering experience with servomechanisms, and on the progress made in cybernetics and in social sciences (see [Richardson 91] for a comprehensive historical account of the emergence of the approach[1]).

[1] For a comprehensive introduction into the general program for cybernetics see [Wiener 48]; the first notion of a servomechanism incorporating human links can be found in [Goodwin 51].

The conceptual framework adopted by system dynamicists has been recently defined as s*ystems thinking* [Senge 90]. To explain how systems thinking influences the handling of problems, let us take an example.

Asked for the reasons why a R&D project has overrun its schedule, most people would tend to provide a list of answers such as:

- the project size has been underestimated
- the workforce productivity has been overestimated
- a huge amount of rework had to be performed
- there was too much turnover among developers
- …

Underlying this mental representation is the concept of one-way causality, which appears to be a natural way of reasoning: a set of causes produces one effect. The causal factors are listed independently, and are considered as events that disturb the system.

The first step toward systems thinking is to recognise the presence of feedback mechanisms in the system, the factors being deeply connected through **circular causal relationships**. For example (see Figure 1), if a project is perceived to be late, this will result in schedule pressure. People will tend to work harder, possibly overtime, which will increase their productivity. On the other hand, they may commit more errors, so that the progress is slower than expected. Perceiving that the project becomes more and more late, the manager may react by hiring new staff, which in turn will have several competitive effects back to the progress.

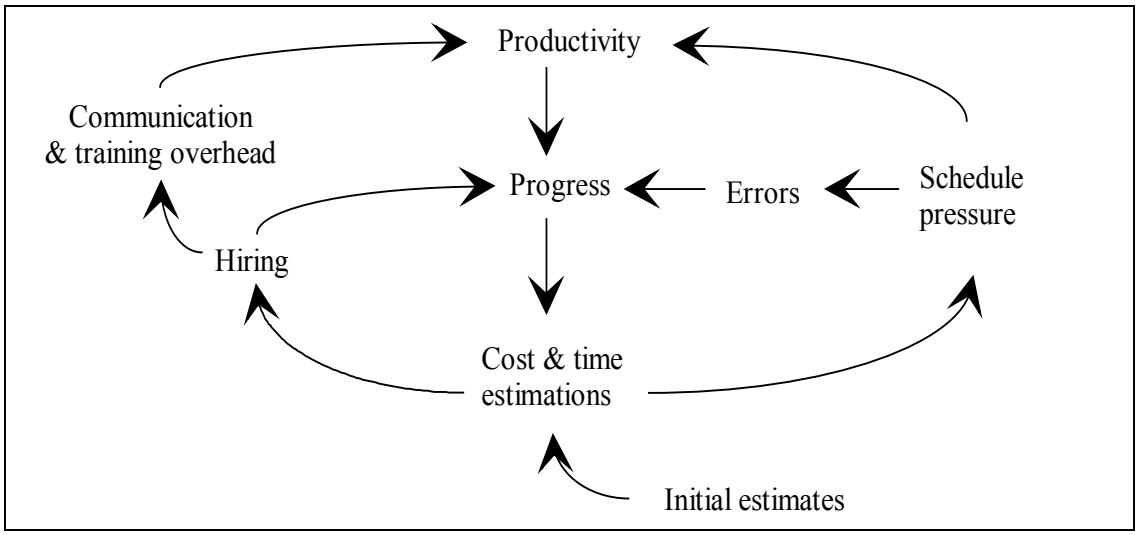

Figure 1: Circular causality in the "late project" problem.

[Senge 90] pointed out that "systems cause their own crises, not external forces or individual's mistakes". In system dynamics the behaviour of a system is considered as primarily being **generated by its structure** – the interaction of all the feedback loops over time. Therefore, in the example above, the project becoming late is diagnosed as originating from endogenous mechanisms, rather than from external disturbances (e.g. initial estimates). The *internal* structure is deemed as the explanatory factor for behaviour[2], and an adequate system dynamics model has to encompass any interaction that is essential to reproduce the behaviour of interest. This also includes the human decision processes which are not modelled on the micro level of individuals but as aggregated flows resulting from a **general policy structure**. For example, the hiring of a new developer arises within

[2] Although socio-economic systems are widely agreed to be 'open' by nature, the discussion conducted in [Richardson 91] shows that some relevant problems may be most profitably tackled by endogenous approaches (like system dynamics).

the framework of a manpower acquisition policy, on whose characteristics (e.g. aggressive policy, or reluctance to change the workforce level, or mix of both depending on the project phase) it is focused. Note here the underlying assumption, central to systems thinking, that people placed in the same structure tend to produce qualitatively similar results.

The management of complex systems, like those typically underlying industrial software development processes, is very difficult. Based only on intuition and experience, it is generally not possible to comprehend the dynamic implications of so many interrelated loops carrying the system structure. If problems occur, their diagnosis is far from trivial. People often fail to think in terms of circular causal relationships and confound symptoms with causes. As a consequence, corrective policies implemented supply poor results for three main reasons: 1) the treatment of symptoms does not suppress the structural cause of the problem; 2) feedback systems resist policy changes because of internal compensation mechanisms; 3) the long term effects may be very different from short term effects, so that the implemented policy may actually worsen the problem in the long run.

The solution advocated by system dynamicists is to **build formal** (mathematical) **models**, and to **perform intensive simulations**. The formal model captures the most important circular mechanisms for the generation of the behaviour patterns of the real system. It necessitates the explicit statement of mental hypotheses, and forms a common basis for the discussion of the problem. A typical model consists of a high-order system of nonlinear differential equations, which cannot be solved analytically. But it can be *simulated* on a computer, so that the behaviour of the system can be studied in a systematic manner. The benefit that can be drawn from simulation is twofold. In the first place, simulation is intended to foster better understanding of the problem dynamics, the supplied behaviour trends being analysed with regard to the internal structure. In the second place, it is hoped that more efficient policies can be found experimentally, by investigating the effect of alternative policy structures. This orientation toward problem understanding and policy analysis is possible only because of the endogenous point of view, which regards the policy structure as part of the model.

III Elements of the system dynamics method

To allow a better understanding of the system dynamics approach, and to provide some technical grounds for the discussion of existing models in the remainder of this report, we briefly outline some basic elements of the method in the following sections.

III.1 Mathematical background

The tutorial published in [Forrester 71] gives a good introduction to the principles of system dynamics and the underlying mathematics; additional material may be found in [Forrester 61, Richardson 81, and Bossel 92]. The dynamic characteristics of feedback systems have been represented in the form of differential equations for a long time. These mathematical models, in the system dynamics approach, are continuous and nonlinear:

- although approximated by discrete step-by-step simulation process, the equations are primarily deemed continuous; the chosen formulation helps to concentrate on the continuously interacting forces in the system, rather than on discrete events arising within this central framework;
- the incorporation of nonlinearities in the equations is deemed critical to match a complex nonlinear reality.

For the purpose of numerical evaluation, the equations are separated into two groups: the **level equations** and the **rate equations**. This terminology of levels and rates is consistent with a flow-structure orientation, and has been introduced by Forrester together with schematic conventions invoking the image of fluid-like processes. A flow diagram (see Figure 2) offers a much more precise pictorial representation of the model structure than does a causal diagram (such as the one in Figure 1): it identifies explicitly the variables chosen to model the system; it distinguishes between integrals and derivatives; it states the parameters involved in each equation.

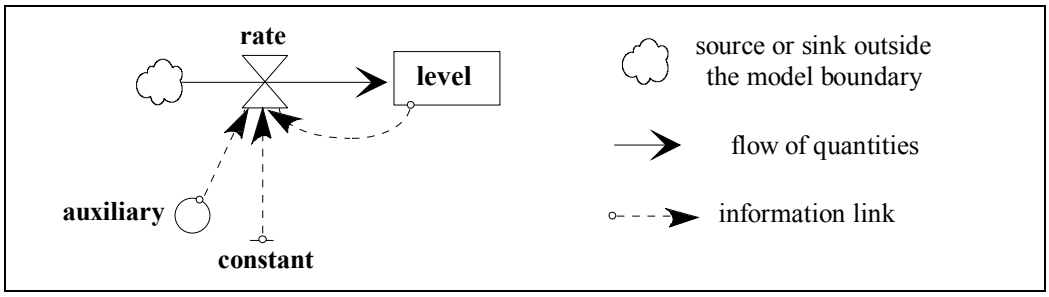

Figure 2: Schematic conventions of flow diagrams.

The **levels** describe the state of the system. Their knowledge at time *t* is sufficient to retrieve the instantaneous value of any other variable. They accumulate (or integrate) the results of action in the systems, an action being always materialised by flows in transit. The derivative of a level, or equivalently the rapidity at which it is changing, depends on its input and output flows. In simulation models, the computation of a level is approximated by a difference equation of the form:

$$\text{Level}(t+dt) = \text{Level}(t) + (\Sigma \text{ input rates} - \Sigma \text{ output rates})\, dt.$$

The **rates** are what change the value of levels. Their equations state how the available information is used in order to generate actions. A rate has four conceptual components: an *observed condition* is compared to a *goal*, and the *discrepancy* found is taken as the basis for *action* (flow) generation. The rate equation that formalises this policy is an algebraic expression that depends only on levels and constant values. **Auxiliary variables** can be used for intermediate computation.

As a concrete example of the use of these notations, let us present a simple mechanism aiming at controlling the number of engineers working on a project:

$$\text{Staff\_level}(t+dt) = \text{Staff\_level}(t) + \text{Hiring\_rate}(t)\, dt$$

The instantaneous hiring rate is given by:

$$\text{Hiring\_rate}(t) = (\text{Indicated}(t) - \text{Staff\_level}(t)) / \text{Adjustment\_time}$$

As the staff level grows at a rate which depends on the previously observed state, we are in the presence of a feedback mechanism. The auxiliary "Indicated" is the goal sought by the loop, which may vary according to some (possibly nonlinear) function not detailed here. "Adjustment_time" is a constant indicating the time needed to find and train a new engineer.

The level and rate variables must alternate along any feedback loop, which can be classified according to three attributes:

- polarity, negative (deviation-counteracting feedback) or positive (deviation-amplifying feedback),
- order (number of levels),
- linearity or nonlinearity (in rate equations).

It is the presence of nonlinear high-order loops in the system that makes the derivation of analytical solutions unfeasible. It can be shown that even simple low-order systems may exhibit a large variety of behaviour patterns; and in practice, any system of interest will involve a high-order, multiple-loop, nonlinear structure.

III.2 The process of modelling

Strictly speaking, there is no formal methodology defined for the construction of dynamic models. However, as more and more experience has been gained in the method, a general procedure has emerged whose steps can be formalised to a certain extent [Randers 80]. To keep the work under control, an **incremental design** is adopted: starting from a model skeleton including only few basic

feedback mechanisms, a series of refinement and test stages is carried out. Some illustrative examples may be found in [Richardson 81, Forrester 61].

The initial step is the definition of the **reference mode** which acts as a catalyst in the transition from general speculation about a problem to an initial model. The reference mode captures the dynamics of the tackled problem, i.e., behaviour patterns and time horizon. It is typically represented by graphs displaying the historical or hypothesised evolution of major system variables over time. Having specified the target aspects of system behaviour in that way, the basic real-world mechanisms producing the reference mode must be identified: a causal diagram with few generic loops is issued.

The implementation of the initial model requires turning this model skeleton into a set of equations. The model variables must be chosen: first the levels, then the rates and finally the auxiliaries. At this point, it is useful to build a flow diagram that visualises the parameter interactions. Then, the rate equations are precisely defined. The model calibration [Graham 80, Hamilton 80] consists of attributing initial values to levels, setting constant parameters, and defining the table functions used to introduce nonlinearities not easily expressed by algebraic functions.

Having implemented the initial model, the incremental **refinement** can start. A refinement step may either extend the model boundary by incorporating new causal mechanisms, or refine the existing structure; some examples are the dissection of a level, the replacement of a constant value by a variable, the addition of a new loop, etc. The refinement process is guided by the need to generate a behaviour mode, to test the effect of a management policy, or to satisfy the client's expectation with respect to realism and predictive accuracy. The refined models will show a variety of behaviour modes, including the reference mode.

Static and dynamic verification techniques are applied throughout all phases of design, and preferably involve both experts in the real system and experts in system dynamics.

As regards **static verification**, the plausibility of any introduced feedback mechanism must be examined, given the knowledge about the real system. The detailed model formulation is inspected in order to verify that it is consistent with loop diagrams. Also, the chosen variables and their initial values have to correspond to quantities that are meaningful in the real system. Some systematic checks are performed on the model equations: dimensional analysis ensures their consistency; rate and auxiliary equations are examined under extreme conditions in order to expose flaws in their formulation. Finally, the model builder must take care that the modification introduced to refine one part of the structure be adequately passed on to other related parts (see the example developed in [Richardson 81]).

After a model version has passed static verification, **dynamic testing** begins. It is verified that the values taken by variables during the simulation remain within their valid range even under extreme conditions. In case of unexpected behaviour modes, the opinion of experts in the real system is requested. **Sensitivity analysis** [Tank-Nielsen 80] is performed, in order to test whether all chosen factors are essential to reproduce a given behaviour mode, and to study the sensitivity of the model with respect to reasonable variations in parameter values or to a reasonable alternative in model formulation. Since it is a powerful means to identify the active and dormant part of the model, sensitivity analysis also gives an insight into influential places for policy implementation.

Once credible model structure and parametrisation is obtained by iteration, it is expected that realistic conclusions can be drawn from **policy experimentation** with the model.

III.3 Languages and Tools

There exist currently a lot of software packages supporting the development and simulation of system dynamics models. The characteristics and features of some of these packages are summarised in Table 1.

Among the used languages, it is worth mentioning Dynamo Plus, developed at MIT, and two more recent graphical languages, Stella and iThink. Two of the most comprehensive integrated

modelling and simulation environments which have been commercially successful are the Vensim and MicroWorld tool families. Both environments do not only offer graphical simulation languages and several useful simulation features, but also allow the building of interactive user interfaces satisfying training needs or supporting on-line decision making processes. Moreover, MicroWorld S**4 can be connected to company owned project data bases and management information systems.

	Vendor	Reference	Hardware	Features
Dynamo Plus	Pugh-Roberts Associates	[Pugh 91]	IBM PC or compatible	Supports only textual simulation language; batch simulation; generation of reports, plots, tables; gaming interface.
Stella **iThink**	High Performance Systems	[Stella 90]	Macintosh (soon: IBM PC or compatible)	Graphical modelling; generation of plots and tables; batch simulation.
MicroWorld Creator	MicroWorlds Inc.	[Diehl 92]	Macintosh (models may be ported to IBM PC)	Supports a textual simulation language, but understands Stella & iThink; classical facilities + design of interfaces for interactive simulations, exploration and playback modes.
MicroWorld S4**	MicroWorlds Inc.	[Diehl 93]	Macintosh (models may be ported to IBM PC)	MicroWorld Creator + understands Dynamo, connects to management information systems, supports large models, provides language to program batch simulations.
Vensim	Ventana Systems Inc.		IBM PC or compatible	Supports a graphical simulation language; provides classical facilities + robustness test; allows structural analysis of the model (causal tracing) and design of interfaces for training situations.
DynaMan	University of Milano	[Barbieri 92]	SUN/VAX	similar to Vensim
Dysmap 2	University of Salford	[Vapenikova 87]	IBM PC or compatible	similar to Vensim

Table 1: Some existing system dynamics software packages.

III.4 Application domains

System dynamics was introduced at MIT to solve industrial problems; hence the original name of **industrial dynamics**. In his reference book [Forrester 61], Forrester gave two detailed examples dealing with cyclical patterns in production, inventory and employment in the manufacturing area. One of them was the first application of system dynamics on a real case, the Sprague Electronic Company (electronic components industry). It was shown that variation in demand of the products was not an adequate explanation for an oscillation: the system structure had the inherent tendency to generate it. Sensitivity analysis was performed to indicate which parts were most crucial in determining this behaviour. A combination of new inventory and employment policies was proposed from model experimentation. The implementation of these policies in the real system, reported in [Fey 78], supplied a mitigated result: during the first months, the situation was improved; however, after about one year, the managers tended to return to their old employment policy and cyclic behaviour patterns appeared again. Note that the structural explanation for long-term cyclic rise and fall of industrial activity has now become a classical result of the method (see e.g. [Randers 80] who mentions a study of the Norwegian pulp industry focused on inventory fluctuation).

Since the early developments at MIT, the method has been applied to model a wider variety of systems, hence the current name of **system dynamics**. A collection of papers published in [Roberts 78a] reports studies from five broad application domains: 1) manufacturing, 2) marketing and distribution, 3) research & development activities, 4) management control and finance, 5) societal problems (ecological and sociological). The most famous of the societal models was the World model [Meadows 72] developed by a research group at the MIT on the initiative of the Club of Rome in the early 70s. It aimed at studying the future of human activity in our finite world, and focused on five physical quantities: population, food production, industrial capital, production and non-renewable natural resources. This work generated much controversy, but gave rise to a new interest in policy modelling efforts in numerous countries.

During the last ten years the **management of software projects** has emerged as a new application domain. The ideas underlying the models of software project dynamics originate mostly from the work previously performed at MIT on R&D projects under the direction of Prof. E. B. Roberts [Roberts 64, 74]. Roberts developed a base model intended to capture the fundamental characteristics of an R&D project and its typical life cycle. The base model is supposed to be adapted to an organisation's specific needs. It is reported in [Roberts 78a] that R&D project models based on the Roberts framework have been implemented by several industrial organisations, among which are Motorola Military Electronics and Sony Labs.

As pointed out in [Abdel-Hamid 91], the management of software projects shares common features and problems with the management of R&D projects, as regards 1) project planning (difficult estimation of the time and money to invest in the project); 2) the control of progress (errors committed between actual and perceived status); and 3) the management of human resources (men and months are not interchangeable); hence the idea that similar causal mechanisms are involved. Indeed, the system dynamics models of software projects incorporate many structural elements of Roberts' model, including [Levary 91]:

- **typical project variables**, such as workforce level, budget, scheduled completion date, number of errors produced, number of errors detected;
- **managerial-related functions**, e.g. staffing, planning and controlling;
- **production-related functions**, e.g. design, development, verification, rework;
- **human-related functions**, e.g. productivity, motivation, error rate, whose values are affected by the project's perceived status (schedule pressure) and the penalty-reward structure of the organisation.

IV Existing system dynamics models for software projects

There are two tendencies observed in the literature: **general models** that try to capture the typical features of a software project, and **specific models** built for the purpose of a particular study. In the first case, the adopted approach is similar to Roberts' approach in R & D: the aim is to provide a base framework for the dynamic analysis of projects, the general model having then to be adapted to the needs of its users. In the second case, the model is directly tailored for a special purpose so that its scope is limited to the problem investigated (see e.g. specific studies conducted in [Chichakly 93]).

Four general models will be presented in this section: the Abdel-Hamid model, SEPS, and two other models. Abdel-Hamid's model has been chosen to pursue our work on software project dynamics issues.

IV.1 Abdel-Hamid and Madnick's model

This model is the one for which we have found the largest amount of material in the literature, and it is currently commercially available. The book [Abdel-Hamid 91] gives a complete description of the model equations and the hypotheses that led to their formulation.

The proposed model integrates the three kinds of functions that influence a project's dynamics: managerial related, production-related and human-related functions. It is confined to the development phase, that is, neither the requirements definition phase, nor the maintenance phase is taken into account: as a result, stable requirements are assumed all along the process. The model focuses on small to medium-scale projects (16-64 KDSI – Delivered Source Instructions).

Building the model required three steps : 1) interviews were conducted with project managers in order to gain understanding of industrial practice – a model skeleton was issued; 2) the model was complemented by information collected in the literature; 3) additional interviews led the authors to revise the model until its final version. During the detailed description of the model in the book, the mathematical hypotheses are stated and discussed according to data gathered during interviews or review of the literature.

The model consists of **four subsystems**, each including several kinds of influential functions: the Human Resource Management subsystem; the Software Production subsystem; the Controlling subsystem; the Planning subsystem. Table 2 outlines their main features. All subsystems are interrelated and interdependent; the local modification of a parameter triggers a chain reaction in the whole system, that will eventually affect the initial parameter.

Human Resource Management		Captures the hiring, training, assimilation and transfer of people. Evaluates the workforce available, newly hired or experienced.
Software Production	**Manpower Allocation**	Evaluates the fraction of manpower allocated to QA, rework, development & test.
	SW Development	Focuses on productivity. Determination of the % of tasks completed.
	QA & Rework	Models the occurrence of errors in tasks, their partial detection (QA) and imperfect removal (rework). Evaluates the number of errors that will pass into subsequent phases of development.
	System Testing	Models the propagation of residual errors into subsequent phases of development, and their elimination by testing. Determines the % of tasks tested.
Controlling		Measures perceived progress and adjusts the job size estimates.
Planning		Adjusts schedule & workforce level according to the estimates.

Table 2: Overview of the model's four subsystems.

A **prototype of about 300 equations** has been implemented in the Dynamo programming language. It has been validated on a real case-study, a completed NASA software project of size 24.4 KDSI. The validation proceeded as follows: 1) the model was run calibrated with NASA data and 2) simulated and actual outcomes (evolution of scheduled completion date, of estimated cost, of manpower loading, of cumulative man-day expenditure) were compared. It emerged that the model is able to reproduce correctly the evolution *patterns*, but the results are less accurate as regards the numerical *values*. This was explained by the fact that – due to tight schedule constraints – the managers were less reluctant than assumed in the model to hire new people during the final stages of the project: this accounts for the slight underestimation of manpower level, which in turn affected the accuracy of cost and duration predictions.

Batch simulations of the model have been performed, in order to gain insight into the general process of software development, and to investigate the impact of some managerial policies on typical example projects. Experimental results can be found in [Abdel-Hamid 90, 91, 93a, 93b, 93c].

To conclude, the work of Abdel-Hamid and Madnick brings a significant contribution to the field. The fact that the model has been entirely published and thoroughly commented on makes it an adequate starting point for our further investigation. Incidentally, it is worth noting that it has already been used by other authors as the basis for their studies: [Aranda 93] mentions an extension of Abdel-Hamid's model with a longer time horizon, covering the successive software releases as well as the market diffusion.

IV.2 SEPS

The Software Engineering Process Simulation (SEPS) has been developed at the Jet Propulsion Laboratory, California Institute of Technology [Levary 91, Lin 93]. It is an extension of an earlier project, the Software Life Cycle Simulator [Lin 89]. To our knowledge, the model equations are not available in the public domain.

The core dynamics model offers a finer description of the software development process than does Abdel-Hamid and Madnick's: it separates the process into detailed phases such as initiation, requirements, preliminary design, detailed design, etc. It is thus more oriented toward the analysis of large projects. Also, it integrates the possibility of introducing requirement changes (but it does not account for successive releases or versions of the software product). Managerial related, production-related and human-related functions seem to be represented with a higher level of detail than in Abdel-Hamid and Madnick's model. As a result, the source code is more complex, involving about **700 Dynamo equations**. The model has been validated against real data from a completed space shuttle software project, covering a time horizon of about 8 years: the supplied results were accurate within a range of 10% of errors. Recently, a statistical analysis has been conducted to test the hypothesis that experienced managers were not able to distinguish between genuine and simulated data [Lin 93].

The salient feature of SEPS is the presence of **input and output expert systems having fuzzy logic** at the interface of the dynamic model. The fuzzy logic has been introduced to handle imprecise information such as subjective input variables (e.g. effectiveness of training methods) or some delay variables (e.g. length of time during which work pressure is applied). The *input expert system* checks the plausibility of input values, performs preliminary analysis using conditional statements, and transforms the fuzzy values into numerical ones that will be used as inputs to the system dynamics model. The purpose of the *output expert system* is to make recommendations based on experimentation with the model; it supports the analysis of output variables using fuzzy algorithms.

IV.3 Other models

We have currently little information as regards the model developed at Draper Laboratory [Smith 93]. The existing prototype has been implemented using the Stella simulation language and the MicroWorld Creator interactive simulation tool (see Subsection III.3). It is primarily intended to serve as a "flight simulator" for managers, hence the interactive interface. The Draper model seems to incorporate classical dynamic assumptions and to be of medium complexity (it is probably more similar in size to the Abdel-Hamid model than to SEPS).

In the framework of their consultant activities, Pugh-Roberts Associates has developed a general model emphasising the rework cycles in projects [Cooper 93]. The model includes several thousands of equations, and has been used to analyse defence and commercial software development projects.

V Potentialities of the approach

The use of system dynamics models offers several interesting perspectives in the field of software project management and software process modelling: five possible directions are proposed in the literature that will be developed in the following paragraphs of this section. Most of them can be related to classical uses of the system dynamics method; hence, their potential or limitation will be discussed according to past experience acquired in other application domains. Table 3 summarises the uses of system dynamics discussed in this section.

System dynamics usage domain	System dynamics usage in classical application domains (not software related)	System dynamics work done in the software field
Research	System dynamics models have proven useful due to their degree of formality and explanatory power. Also, simulations based on system dynamics models allow controlled experiments on non-physical systems.	Experimental results have been published.
Training	Flight-simulator-type modelling and simulation environments have been used for the training of systems thinking and the confrontation of managers and politicians with realistic situations.	Some interactive prototypes exist.
Policy investigation	Policy investigation has been the vocation of system dynamics. Models mainly support communication among managers and facilitate diagnoses. Simulations are applied for policy experimentation.	Case studies are published, either performed by managers (small models supporting strategic decision process) or by consultants (large scale studies).
Post-mortem analysis	*No information available.*	There are some hints on how to learn from simulation of completed projects.
Monitoring ongoing projects	Whether system dynamics models possess adequate numerical accuracy is debatable.	This usage is subject to prospective research.

Table 3: Potential uses of the system dynamics method.

V.1 Research – toward a theory of software project management

System dynamics models possess a high explanatory power, due to their feedback structure, and they are simulation models. They are intended to be both a formal statement of some knowledge on the target system *and* a source of new knowledge: researchers can use them as tools for learning about the process of software development.

System dynamics models are formal. Of course, being formal does not imply being accurate with respect to reality; but at least, the formal representation is a convenient support for discussion, criticism and suggestion for improvement. The process of model building in itself improves researchers' understanding of the system. It forces them to **develop systems thinking**, to **identify the key parameters** that influence the system behaviour, and to **state precise hypotheses** about the way these parameters interact. As argued in [Forrester 71], "the validity and usefulness of dynamic models should be judged not against an imaginary perfection, but in comparison with the mental and descriptive models which we would otherwise use".

Model simulations make it possible to study the dynamic implications of many interrelated loops, which lay beyond the scope of human intuition. It becomes possible to **test dynamic hypotheses** for an observed behaviour mode, and to perform **controlled experiments**. In such experiments, the isolation of effects is ensured by simulating and analysing project variants that differ from a (reference) base case in only few relevant parameters; hence facilitating a systematic analysis of experimental results. This potentiality offered by the system dynamics method was early stated in the reference book [Forrester 61]: "by using a model of a complex system, more can be learned about internal interaction than would be ever possible through manipulation of the real system". Indeed, **experimentation on real industrial software projects would not be realistic**, because of the waste of time and money that it would imply; moreover, the experimenter would never achieve the ideal conditions for the isolation of effects, for lack of total control on the real world.

Controlled experimentation with software project models is exemplified in [Abdel-Hamid 91] on a chosen base case medium in size: the initial project estimates are derived from COCOMO [Boehm 81] for 64 KDSI assumed; the other model parameters are set to typical values reported in the literature. The performed experiments can be classified according to the goal pursued:

- **Dynamic hypothesis testing.** The authors presumed that the managerial characteristics of a company should have a significant impact on project cost. Experimental results corroborate this conjecture for the following parameter changes: 1) staffing policy (people allocated partial instead of full time on the project); 2) willingness/reluctance to change workforce level; 3) effort distribution between development and testing.
- **Study of patterns often exhibited by projects**. The 90% syndrome problem is investigated: schedule overruns usually become apparent when 80-90 % of the planned work is completed. Simulations expose this situation most acutely when the staff productivity is overestimated. Experiments on Brooks' law, which states that "adding manpower to a late project makes it later", lead the authors to conclude that the law does not apply to medium sized projects.
- **Study of managerial techniques**. Investigations of the efficiency of estimation techniques involved 1) a policy consisting in multiplying the initial estimates by a safety factor and 2) a technique of estimation by analogy (experimented by conducting successive simulations, taking each time the result of the previous run as the initial estimate). It is observed that any project tends to consume all its planned resources, even if oversized.
- **Search for process optimisation**. Intensive simulations over a parameter range make it possible to find empirically an optimal value for this parameter in the target project: this is exemplified in the parameter representing the amount of effort allocated to QA.

Other experiments on the Abdel-Hamid model, not reported here, also deal with project estimates [Abdel-Hamid 93a, 93b, 93c]; the effect of schedule compression is investigated in [Abdel-Hamid 90]. At Draper Laboratory, simulations have been performed to study the cost of errors made in the early phases of software development; general advice for the management of human resources are also derived experimentally [Smith 93].

V.2. Training – flight-simulator-type environment

The potential of simulation models for the training of managers has long been recognised: flight-simulator-type environments (or *microworlds*) confront managers with realistic situations that they may encounter in practice, and allow them to **develop experience** without the risks incurred in the real world. The "fun" aspect of role playing and microworld exploration makes the learning process more attractive. Two detailed examples of training workshops based on real industrial cases can be found in [Graham 92] (other examples are mentioned in [Morecroft 88]):

- **People Express Management Flight Simulator**. The game is based on a system dynamics model of *People Express Airlines*, a real company in the 80s that ran through a phase of

spectacular success, followed by a phase of sharp decline and finally failure. The players take the role of the top manager of the company, and experience problems linked to the management of growth, trying to avoid the deregulation observed in the real case.

- **Intecom PBX**. The simulation model describes the entry of a new company, Intecom, into the market of PBXs (telephone switching systems) in a context of changing technology (from electromechanical to electronic PBXs). The players have to manage properly the transition from the old to the new technology to obtain a high market share.

As regards the specific topic of software project management, experimental studies have been conducted on existing model prototypes.

The experiments carried out at the Jet Propulsion Laboratory aimed at studying the decision-making process of software managers, in order to identify strategic information [Lin 93]. Twenty managers were asked to conduct a project simulated with the aid of SEPS; some were provided with cause-effect feedback of their actions, while the others were not. It was observed that the second group (without feedback information) tended to act on a more "fire fighting" way than the first one; and the feedback information was most beneficial to the less experienced managers.

The Draper model served as the basis for an experiment involving a group of 50 experienced managers [Smith 93]. The scenario of the game involved a 15 percent requirement change in the course of the simulated project. Few of the managers were able to adapt properly to this situation: most of them reacted by hiring new staff late on the project (a "fire fighting" policy), and experienced budget and schedule overruns. Worse, the managers tended to reproduce *exactly the same errors* when running the scenario for the fourth or fifth time. The authors conclude that the managers were limited by their mental model of the process, but that they were reluctant to change it.

The two experiments reported above show that the natural one-way causal thinking can be detrimental to the efficiency of managers. In the opinion of system dynamicists, the aim of the training should go beyond that of facing people to realistic problems; the concern is also to make managers **adopt systems thinking,** and perceive the existence of circular causal relationships. This sets the problem of an adequate game interface and more importantly of the workshop organisation: just running a simulation model as a black box may not be sufficient to have people gain insight into the software development process, and accept to alter their mental model. The definition of effective training environments is an important field of research for system dynamics. As an example, we will describe how the workshops based on *People Express Management Flight Simulator* and *Intecom PBX* are organised.

Both workshops begin with a short presentation of the case study (respectively People Express Airlines and the PBX market); then, they proceed as follows:

People Express	**Intecom PBX**
• General discussion on the case study; search for adequate strategies.	• Case discussion supported by Stella diagrams; simulation may be used to answer questions.
• Rapid presentation of the model and its structure; the game is run several consecutive times by groups of two or three persons.	• Divided in small teams, the participants have to define their strategy, justify it, and predict its results.
• In-depth discussion of the obtained results, supported by simulation of proposed strategies and hypotheses.	• The teams have to present their strategy to the others; the instructor simulates them on-line.

In both of these workshops, the participants are provided with a **white-box view** of the case study; the circular relationships that are hypothesised are explained to them. During the game, they are encouraged to **adopt a scientific approach**: in the People Express case, they have the possibility to systematically experiment their strategy by simulation; in the Intecom PBX case, they have to document their decision and forecast its effect in a detailed manner. The **discussion of results** is then a critically important part of the workshop.

Such a type of course organisation avoids the situation observed at Draper Laboratory, where managers played the game without analysing their errors, hence gaining little insight into the problem. The participants are expected to learn the concepts of systems thinking and to use systematic exploration to cope with the counterintuitive behaviour of feedback systems.

We believe that flight-simulator-like environments based on system dynamics models have the potential to **play a central role in management training**, because of the conceptual tools they bring into this matter. Ideally, the expected long term effect of training is to have managers relate generic feedback patterns to common industrial problems, and use simulation models to investigate new policies. Training is thus a prerequisite for the acceptance by managers of system dynamics as a support for their strategic decisions, which, in our opinion, constitutes the most promising potentiality of the method.

V.3 Strategic decisions – evaluation of alternative policies

Policy investigation is the vocation of software dynamics. A system dynamics model can be used as an electronic laboratory where hypotheses about observed problems can be tested, and corrective policies can be experimented before they are implemented on the real system. Significant benefits can be drawn from:

- introducing the use of simulation in the meetings of policy makers. The simulation is a tool to investigate on-line the suggestions that arise during the meeting. More importantly, the formalism of the model should allow policy makers to share their views in an unambiguous way and harmonise their mental models of the process. Hence, system dynamics models provide a convenient support for executive dialogue.
- performing studies of systemic problems, leading to new policy recommendations in the scope of consulting activities.

Both items suppose that system dynamics has gained acceptance by top management: in the first case, managers have integrated the method as a support for discussion; in the second case, they accept the risk of implementing the policy recommended at the end of the study. In this context, past experience of consultants using the system dynamics method has shown that obtaining credibility is a critical point. One way to achieve this is by training managers and project leaders to the method, so that they are more prepared to accept the principle of simulation.

In the literature of software project management it has likewise been reported that system dynamics supports discussion between policy makers. The formality of the models constitutes a shared language that should facilitate communication, especially if graphical modelling tools are used (cf. Subsection III.3). [Aranda 93] emphasised the importance of **achieving harmony among the mental models of different organisational functions**. During meetings, what-if analyses can be conducted by simulation as new hypotheses or policies are suggested. To serve the purpose of communication, the models have to be kept transparent, that is, they must be of small complexity: this is achieved either by building small models devoted to the analysis of the specific problem, or by using a subset of a complete model of the system, feedback loops being cut if they are not immediately of concern. Such *partial* models have been proven useful to generate plausible scenarios policymakers can identify with [Morecroft 85, Sterman 85, Morecroft 88]. [Chichackly 93] reports that small system dynamics models have been used by High Performance Systems, Inc., to help analyse real-case problems:

- **Introduction of a new technology**, namely object-oriented design and programming. The model was intended to help manage the transition, and incorporated both long-term effects of the technology (intensive code reuse) and short term ones (inexperience of developers, loss of existing code base of past products, use of new tools).
- **Quality improvement**. The model was focused on the software development life-cycle (specification, design and coding), and aimed at optimising quality initiatives: the

introduction of techniques such as reviews and prototyping was simulated at several project stages.

If small models are useful to support executive dialogues, **large-scale studies of systemic problems** involve more complex models, and generally necessitate the intervention of external consultants having a high level of expertise in system dynamics. **The problem is then to have the method gain acceptance of top managers**, since they will be in charge of implementing the policies suggested by the experts. As mentioned in Subsection III.4 the first real-case application of system dynamics was not fully successful: after about one year, the managers tended to return to their old policy.

The experience of Pugh-Roberts Associates, a consultant company specialised in system dynamics, illuminates this problem. [Weil 80] reports on three case studies that illustrate the evolution of their approach to policy implementation, stressing the **ever growing concern of intensive client involvement** during all phases of modelling and policy investigation. The lessons drawn from past failures led to the following conclusions:

- The client contact must be a top line manager, not a staff person.
- The client must be closely involved in the model conception. It is expected to be a technical contribution, which implies that the client receives training on the system dynamics method, and accepts to devote significant effort to model development and validation (in early studies, the client was only considered as a source of information).
- To gain credibility, large models (thousands of equations) are built. Emphasis is put on numerical accuracy.
- The transfer of technology is a big concern. The end product of the study includes detailed policy recommendations and the model is left to the client with technical documentation. Continuing strategic consultation after the final report is deemed necessary to have policy recommendations properly implemented.

Pugh-Roberts Associates now claim to have been successful in improving dozens of major development programs [Cooper 93].

V.4 Post-mortem analysis – learning from past projects

Analysing a completed project is a common means for organisations to learn from past experience, and to improve their software development process. System dynamics can facilitate post-mortem analysis: by properly calibrating a model, it becomes possible to **replay the project, diagnose management errors which arose, and investigate policies** that would have supplied better results.

As an example, simulations of SEPS on a case study indicated that a completed project could have met its schedule at lower costs if the initial estimate had been 15 percent lower [Lin 93]. This was traced to the fact that the staff productivity had been underestimated, being inadequately based on a previous phase involving training to new tools: this led to overstaffing, with additional costs for training, communication, and errors.

Abdel-Hamid also mentions the problem of overestimation [Abdel-Hamid 93c], and points out that estimation by analogy might be misleading and supplies, in the long run, a permanent degradation of performance. This was illustrated in [Abdel-Hamid 91] by the following experiment: the *same* project was run several times, but with different initial estimates each time based on the previous simulation result, and it was observed that the project consumed larger and larger resources. Actually, this phenomenon corresponds to a generic systemic problem known as *goal erosion*, also encountered in other application domains[3]. To avoid having the organisation reproduce – and amplify – its past errors, [Abdel-Hamid 93a] recommends finding optimal values of past projects by simulation, and recording these values for future estimation, instead of recording real values that reflect inefficient

[3] See, for example, the systems archetypes listed in [Senge 90].

policies. Note that this implies that the numerical values supplied by the model can be relied upon, which is quite questionable. The same problem will arise in the following subsection, where we will consider the possibility to base operational decisions on model estimates.

V.5 Monitoring projects – continuous planning and control activities

Can system dynamics models be used to track on-going projects? The question is highly debatable. For the analysis of general behaviour patterns, a qualitative adequacy of the simulation results may be sufficient; but when it comes to taking decisions for continuous planning and control activities, a high numerical adequacy is required.

The question refers to the problem of the predictive accuracy of system dynamics models; and this has long generated controversy: some authors have argued that numerical precision is an irrelevant goal, due to the noise inherently present in systems involving human beings; others claim being able to obtain accuracy within a range of 10 percent [Weil 80]. Of course, this is at the expense of building complex models, and it is clear that such models involving thousands of equations can only be handled by people having a high level of expertise in system dynamics.

In the software project literature, the expressed opinions strongly diverge. [Aranda 93] writes: "microworlds are appropriately used for training rather than for operational decision, because such models describe general rather than precise behaviours". But [Levary 91, Abdel-Hamid 93a] mention the monitoring of projects as a beneficial use of system dynamics models.

For the **initial planning**, some well-known estimation tools (e.g. COCOMO) do not allow the taking into account of factors such as resource availability or schedule constraints. Simulations can then be conducted to adapt the estimates to organisational realities. Abdel-Hamid presents an example where the plan has to be modified to cope with a temporary staff shortage: various scenarios are suggested and evaluated on the model. The author suggests that it should also be possible to continuously react to situations occurring *during the project development*: cost/schedule trade-off analyses can be conducted through simulation, in order to **make mid-project corrections**. However, the model parameters have to be adjusted according to feedback data on the project, in order to have the model reflect the current state; and the problem of model calibration is far from trivial. A possible direction to enhance the model's continuous estimation-correcting capability resides in the synthesis of managerial judgement with statistical methods. In the current state-of-the-art, this field is still prospective research.

VI Conclusion

Reviewing the literature and relating the maturity of the system dynamics approach in the five directions outlined above to the potential soundness and usefulness in the field of software development, we found that the most interesting potentialities reside in research, training, and policy experimentation.

Based on the synthesis of the state-of-the-art of the system dynamics approach, on the investigation of system dynamics tools, and on the comparison with other modelling paradigms not reported here, namely classical statistical approaches (e.g. [Putnam 78]) and more recent formal (e.g. [Klingler 92]) or object-oriented/rule-based approaches (e.g. [Ludewig 92]), our future research on software project modelling will emerge from the previous work of Abdel-Hamid and Madnick. This is mainly due to the fact that we consider their exemplification of how to apply the system dynamics approach on software project management most appropriate with respect to both the level of abstraction of their model as well as its explanatory power. Of course, their model has to be adapted to the needs of large-scale software projects. As one step into this direction, Abdel-Hamid and Madnick suggested to divide the software development part of their software production subsystem (cf. Subsection IV.1) into finer phases (e.g., preliminary design, detailed design, etc.) separated by formal milestones. This is in accordance with industrial practice for software production where waterfall oriented life-cycle models are still very common. In a later step, the system dynamics model

should be further extended to a multiple project model or to a model that allows the coverage of several product releases with overlapping development cycles. Other weaknesses of the current model are the assumption of stable requirements during the development process, and the lack of measures for software quality. It would be useful to discuss the assumptions of the model – and its desirable extensions – with experienced managers, possibly in the framework of interviews.

The validity of the model will have to be tested against real data from completed projects. The extent to which the model reproduces the qualitative and quantitative behaviour patterns will be investigated. This implies that we determine precisely what should be collected to: 1) calibrate the model; 2) compare the results of simulation with data from the real projects.

Further research could be directed at the extension of the system dynamics approach by including hierarchical modelling techniques [Möhring 92]. This would allow, for example, the adequate mapping of hierarchical team structures including the dynamics of cross-level information flow.

More directly related to industrial concerns are the issues of training and policy experimentation:

- As regards **managerial education**, there has been little work specifically devoted to software project models. However, the design of effective learning environments has reached a rather high level of maturity in other managerial applications, and insight should be easily portable to the field.
- As regards **policy experimentation**, the beneficial use of system dynamics has already been demonstrated on real-case studies.

It is worth noting that the two issues are closely related: the training contributes to spreading the concepts underlying system dynamics and prepares managers and project leaders for the acceptance of policy recommendations issued from simulation. On the other hand, interest in training is all the higher if concrete experience has proved the usefulness of the method to support strategic decision. Therefore, both directions, training as well as decision support, should be developed simultaneously.

VII Acknowledgements

The work presented above was supported by Siemens AG. We would like to thank Walter Lamprecht, Karl Lebsanft, Peter Liggesmeyer, Beate Roessler and Simon White for their valuable comments on earlier versions of this paper.

References

[Abdel-Hamid 90] T.K. Abdel-Hamid, "Investigating the Cost/Schedule Trade-Off in Software Development", *IEEE Software*, pp. 97-105, Jan. 1990.

[Abdel-Hamid 91] T.K. Abdel-Hamid, S.E. Madnick, *Software Projects Dynamics – an Integrated Approach*, Prentice-Hall, 1991.

[Abdel-Hamid 93a] T.K. Abdel-Hamid, "Adapting, Correcting and Perfecting Software Estimates: a Maintenance Metaphor", *IEEE Computer*, pp. 20-29, March 1993.

[Abdel-Hamid 93b] T.K. Abdel-Hamid, "Thinking in Circles", *American Programmer*, pp. 3-9, May 1993.

[Abdel-Hamid 93c] T.K. Abdel-Hamid, K. Sengupta, D. Ronan, "Software Project Control: An Experimental Investigation of Judgement with Fallible Information", *IEEE Trans. on Software Engineering*, pp. 603-612, Vol. 19, No. 6, June 93.

[Aranda 93] R.R. Aranda, T. Fiddaman, R. Oliva, "Quality Microworlds: modeling the impact of quality initiatives over the software product life cycle", *American Programmer*, pp. 52-61, May 1993.

[Barbieri 92] A. Barbieri, A. Fuggetta, L. Lavazza, M. Tagliavini, "DynaMan: a Tool to Improve Software Process Management through Dynamic Simulation", *Proceedings. Fifth International Workshop on Computer-Aided Software Engineering*, Montreal, 6-10 July 1992.

[Boehm 81] B.W. Boehm, *Software Engineering Economics*, Prentice Hall, 1981.

[Bossel 92] H. Bossel, *Modellbildung und Simulation*, Vieweg, Braunschweig/Wiesbaden, 1992.

[Chichakly 93] K.J. Chichakly, "The Bifocal Vantage Point: Managing Software Projects from a Systems Thinking Perspective", *American Programmer*, pp. 18-25, May 1993.

[Cooper 93] K.G. Cooper, T.W. Mullen, "Swords and Plowshares: The Rework Cycles of Defense and Commercial Software Development Projects", *American Programmer*, pp. 41-51, May 1993.

[Diehl 92] E.W. Diehl, *MicroWorld Creator User's Guide,* MicroWorlds, Inc., Cambridge, MA, 1992.

[Diehl 93] E.W. Diehl, "The analytical Lens Strategy-Support Software to Enhance executive Dialog and Debate", *American Programmer*, pp. 26-32, May 1993.

[Fey 78] W.R. Fey, "An Industrial Dynamics Case Study", in *Managerial Applications of System Dynamics*, Roberts (Ed.), the MIT Press, pp. 117-138, 1978.

[Forrester 61] J.W. Forrester, *Industrial Dynamics*, Productivity Press, Cambridge, 1961.

[Forrester 71] J.W. Forrester, *Principles of Systems*, Productivity Press, Cambridge, 1971.

[Goodwin 51] R.M. Goodwin, "Econometrics in Business-Cycle Analysis", in *Business Cycles and National Income*, A.H. Hansen Ed., W.W. Norton and Co., New York, 1951.

[Graham 80] A.K. Graham, "Parameter Estimation in System Dynamics Modeling", in *Elements of the system dynamics method*, J. Randers Ed., Productivity Press, Cambridge, pp. 143-161, 1980.

[Graham 92] A.K. Graham et al., "Model-supported case studies for management education", *European Journal of Operational Research* 59, pp. 151-166, 1992.

[Hamilton 80] M.S. Hamilton, "Estimating Lengths and Orders of Delays in System Dynamics Models", in *Elements of the system dynamics method*, J. Randers Ed., Productivity Press, Cambridge, pp. 162-183, 1980.

[Klingler 92] C.D.. Klingler et al., "A Case Study In Process Representation Using MVP-L", *Proc. IEEE COMPASS*, 1992.

[Levary 91] R.R. Levary, C.Y. Lin, "Modelling the Software Development Process Using an Expert Simulation System Having Fuzzy Logic", *Software – Practice and Experience*, pp. 133-148, Feb. 1991.

[Lin 89] C.Y. Lin, "Computer-Aided Software Development Process Design", *IEEE Trans. on Software Engineering*, pp. 1025-1037, Sept. 1989.

[Lin 93] C.Y. Lin, "Walking on Battlefields: Tools for Strategic Software Management", *American Programmer*, pp. 33-40, May 1993.

[Ludewig 92] J. Ludewig et al., "SESAM – Simulating Software Projects", *Proceedings of the Software Engineering and Knowledge Engineering (SEKE) Conference*, Capri, Italy, 1992.

[Meadows 72] D.H. Meadows et al., *The Limits to Growth – a Report for the Club of Rome's Project on the Predicament of Mankind*, Universe Books, New York, 1972.

[Möhring 92] M. Möhring, V. Strotmann, A. Flache: *MIMOSE - Einführung in die Modellierung, Sprachbeschreibung*, Koblenz, April 1992.

[Morecroft 85] J.D.W. Morecroft, "Rationality in the analysis of behavioral simulation models", *Management Science*, 31/7, pp. 900-916, 1985.

[Morecroft 88] J.D.W. Morecroft, "System dynamics and microworlds for policymakers", *European Journal of Operational Research* 35, pp. 301-320, 1988.

[Putnam 78] L.H. Putnam, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem", *IEEE Trans. on Software Engineering*, pp. 345-361, July 1978.

[Pugh 91] *Professional DYNAMO Plus Reference Manual*, Pugh-Roberts Associates, Cambridge, 1991.

[Randers 80] J. Randers, "Guidelines for Model Conceptualization", in *Elements of the system dynamics method*, J. Randers (Ed.), Productivity Press, Cambridge, pp. 117-139, 1980.

[Richardson 81] G.P. Richardson, G.L. Pugh, *Introduction to System Dynamics Modeling and Dynamo*, MIT Press, Cambridge, 1981.

[Richardson 91] G.P. Richardson, *Feedback Thought in Social Science and Systems Theory*, University of Pennsylvania Press, 1990.

[Roberts 64] E.B. Roberts, "Research and Development Policy Making", *Technology Review* 66, no. 8, pp. 3-7, June 1964. Also reprinted in *Managerial Applications of System Dynamics*, Roberts (Ed.), MIT Press, Cambridge, pp. 283-292, 1978.

[Roberts 74] E.B. Roberts, "A Simple Model of R & D Project Dynamics", *R&D Management*, vol. 5, no. 1, October 1974. Also reprinted in *Managerial Applications of System Dynamics*, Roberts (Ed.), MIT Press, Cambridge, pp. 293-314, 1978.

[Roberts 78a] E.B. Roberts (Ed.), *Managerial Applications of System Dynamics*, MIT Press, Cambridge, 1978.

[Roberts 78b] E.B. Roberts, "Systems Dynamics – An Introduction", in *Managerial Applications of System Dynamics*, E.B. Roberts (Ed.), MIT Press, Cambridge, pp. 3-36, 1978.

[SEI 91] Software Engineering Institute (SEI), *Capability Maturity Model*, Technical report CMU/SEI-91-TR-24, Carnegie Mellon University (CMU), 1991.

[Senge 90] P.M. Senge, *The Fifth Discipline – the Art & Practice of the Learning Organization*, New York: Doubleday, 1990.

[Smith 93] B.J. Smith et al., "Death of a Software Manager: How to Avoid Career Suicide through Dynamic Software Process Modeling", *American Programmer*, pp. 10-17, May 1993.

[Stella 90] *Stella II User's Guide*, High Performance Systems Inc, Hanover, 1990.

[Sterman 85] J.D. Sterman, "A Behavioral Model of the Economic Long Wave", *Journal of Economic Behavior and Organization*, 6/1, pp. 17-53, 1985.

[Tank-Nielsen 80] C. Tank-Nielsen, "Sensitivity Analysis in System Dynamics", in *Elements of the system dynamics method*, J. Randers (Ed.), Productivity Press, Cambridge, pp. 187-204, 1980.

[Vapenikova 87] O. Vapenikova, B. Dangerfield, *DYSMAP2 User Manual*, University of Salford, 1987.

[Weil 80] H.B. Weil, "The Evolution of an Approach for Achieving Implemented Results from System Dynamics Projects", in *Elements of the system dynamics method*, J. Randers (Ed.), Productivity Press, Cambridge, pp. 271-291, 1980.

[Wiener 48] N. Wiener, *Cybernetics*, John Wiley and Sons, New York, 1948.