Spring 5-7-2010

Project dynamics: An analysis of the purpose and value of system dynamics applied to information technology project management

Timothy Charles Delobe
James Madison University

Project Dynamics: An Analysis of the Purpose and Value of System Dynamics Applied to Information Technology Project Management

Timothy Charles Delobe

A thesis submitted to the Graduate Faculty of

JAMES MADISON UNIVERSITY

In

Partial Fulfillment of the Requirements

for the degree of

Master of Science

Integrated Science and Technology

May 2010

Acknowledgements

First off, I would like to thank my thesis advisor, Dr. Geoffrey Egekwu, for all of his guidance, assistance and availability throughout this entire process. His support was invaluable in helping me stay on track and complete the thesis. I would also like to thank my other committee members, Dr. Michael Deaton and Dr. Mohammed Zarrugh for their sincere support and direction, as well as their valuable feedback. Dr. Deaton's courses in System Dynamics provided the foundation for this project and were critical to my success.

In addition, I would like to thank my wife, Meghan Delobe, for her unwavering support over the past two years as I worked towards the completion of both this degree and my thesis. Without her help, none of this would have been possible.

Finally, I would like to thank my newborn son, Charles Andrew Delobe, for providing me with all of the motivation to stay on track and finish this project on time and under budget.

Table of Contents

List of Tables

List of Figures

Abstract

Project failure in the Information Technology (IT) sector is well documented in the literature; project managers miss their target budgets and schedules more than twice as often as they meet them. Traditional project management methodologies initially developed for the large-scale engineering projects of the 1950's, while still relevant and useful, are reductionist in nature and are therefore missing a systems approach that concentrates on knowledge creation before, during and after a project. The research presented herein will demonstrate the role of system dynamics in augmenting a project's control processes, as well as the skill set used by the project manager. Research from a wide variety of projects within the information technology sector will be synthesized, some using system dynamics methodologies, and will serve as the basis to comparatively analyze the value added using this novel project management approach. The project dynamics and lessons learned within will illustrate the complex interactions and feedback structures inherent in all projects, as well as seek to educate project managers on their cause-effect relationships. Furthermore, the research will illustrate problematic project dynamics, using various conceptual models, and suggest the need to integrate system dynamics methodologies for project management into traditional project management processes and bodies of knowledge instead of solely relying on them as a post-mortem tool for project analysis.

I. Introduction

While the discipline of project management has been around since early civilization, it has gained importance in recent decades as a rapidly growing management discipline in a wide variety of public and private organizations. Its modern practice began in the 1950's with large defense, aerospace engineering and construction projects; however, recent trends such as globalization, corporate down-sizing, shorter product life cycles, accelerated technological change and increased market competition have all contributed to what David Cleland, a project management scholar, refers to as the rise of the "Age of Project Management" (Gray and Larson 2000). One particular class of projects (information technology) fail at an alarming rate (Standish Group 2009) and the discipline remains inadequately researched and poorly understood compared to the other more established management disciplines like operations management and new product development. For example, out of the more than 7000 Harvard Business School case studies, only 2% mention the discipline of project management and out of those there are only a few dozen that deal with real project management related issues (Shenhar, Dvir 2007).

Project failures, delays and cost over-runs are all well documented in the literature and adequately illustrate the problems all organizations face with their project related initiatives. According to a Standish Group study (2009), only 32 percent of information technology (IT) projects are successful, 24 percent are failures, and 44 percent were completed but delivered late, over budget and/or without the required scope. In other words, over two-thirds of all IT projects either fail to deliver on the project management plan but still deliver something, or simply never produce a deliverable at all. Some other studies show that upwards of 85 percent of projects, across a wide range of industries, miss their scheduling and budget goals; on average, these failed projects miss their schedule goals by as much as 70 percent and their budget goals by as much as 60 percent (Shenhar and Dvir, 1993). However, 100 and 200 percent cost over runs very common as well (Sterman 1992).

Multiple operational methods have been deployed since the 1950’s to address the need to better manage projects; however, there has been considerable emphasis on methodologies and tools (e.g. PERT, Work Breakdown Structure, Critical Path Method, GANT Charts, Earned Value Analysis) that focus on dissecting projects into their component parts and emphasizing that if each part can be understood and managed the project as a whole can be understood and managed. These tools are focused on providing important and useful methods for operational concerns and project planning; however, some studies suggest that the main causes of project failure lie predominantly in strategic areas such as the political, legal and behavioral aspects of projects (Rodrigues and Bowers 1997).

Moreover, regardless of the increased success of these operational tools, projects still routinely fail and cost organizations in both the private and public sectors $80 - $145 billion per year in the process (Standish Group 2009). These reductionist strategies to project control are useful models; however, they are myopic because they often times fail to properly account for the project’s macro view and the reality that slight perturbations in any of the component parts of a project schedule can have cascading and escalating effects on other work packages and the project as a whole; thereby causing increased error rates and the potential to miss schedule and budget targets. Poor project success rates suggest the need to not only better theoretically define and understand the discipline, but to also reexamine the standard methodologies used and develop new strategies to modify, replace or augment those traditional approaches.

System Dynamics is one area of research that has begun to tackle this problem from the strategic perspective and aid in management's ability to understand complex socio-technical systems. It is a methodology used by both industry and academia to model, simulate and manage complex feedback mechanisms (System Dynamics Society 2009). Such mechanisms are universally present in large IT projects and, as the literature suggests, often the cause of project failure. Much like other systems approaches to problem solving, system dynamics focuses more attention on the higher level dynamics (i.e. feedback) that exists between project components. These dynamics can often produce counter-intuitive behavior and directly affect project success rates. Within a system dynamics framework, less attention is paid to traditional Project Management Institute (PMI) models and methodologies such as Program Evaluation and Review Technique (PERT), Work Breakdown Structure (WBS), GANT charts and Earned Value Analysis (EVA), which serve as complimentary toolsets, but are considered to be more detailed operational methods for project management.

The goal for this paper is to present a comparative analysis between traditional models and methods of project management and the recent developments in the use of system dynamics to augment the more traditional operational approaches with a higher level strategic understanding of how certain project dynamics affect project outcomes. A review of the project management and system dynamics literature will serve as the basis for the research and provide the foundation to draw conclusions about system dynamics' value to the project management body of knowledge. The intent here is not to build a running quantitative system dynamics model, but rather synthesize existing research and models, into an integrated framework of lessons learned that will help project managers in the field better understand the dynamic and complex nature of projects and the purpose and value of the system dynamics methodology. Based on this research, suggestions for future research will be presented, such as the development of quantitative models that are dynamic in nature and integrated into the traditional PMI body of knowledge and project management framework. The paper will explore the possibility of integrating system dynamics models into the project management process instead of the more traditional method of using them strictly as a post-mortem performance analysis tool.

II. The Problem

a. Current State of Project Performance

Contemporary business processes are more complex than in decades past and resemble structures with more interrelationships and interdependencies. It is becoming clear that the conventional (i.e. reductionist) project management methods are inadequate for handling this new era of complexity. The Project Management Institute recognizes this trend and research is currently underway by PMI, as well as others in academia, to learn how best to manage complex projects (Haas 2009). Management styles and organizational structures are also following this trend towards complexity. Historically, managers subscribed to a Newtonian philosophy of management resembling a machine model predicated on linear thinking, control theory and predictability; this model is proving very difficult to tender in the new era of complexity (Lewis 2008, Hart 2009, Haas 2009). Some executives are even rejecting traditional organizational structures in favor of more complex models, like the matrix organization, or another type that Haas (2009) refers to as “alliances,” whereby an organization creates an organizational structure comprised of complex interrelationships between suppliers, partners, regulatory entities, customers and sometimes even competitors. Out of the complex organizations, complex projects are born.

The project management literature over the last three decades suggests a poor performance record for complex IT projects; particularly with regards to software development projects (Brooks 1975, Abdel-Hamid and Madnick 1991, Rodrigues 1996, Dorsey 2000, Lewis 2008, Haas 2009, Hart 2009). Some of the literature following this trend references *The CHAOS Report* by the Standish Group (2009), which is considered to be the most comprehensive, ground-breaking study to-date on information technology related project performance. Since its first release in 1994, the Standish Group has followed the IT industry's performance, or lack thereof, and has regularly updated *The CHAOS REPORT* roughly every two years.

Year	**Successful Projects**	**Failed Projects**	**Challenged Projects**
2008	32%	24%	44%
2006	35%	19%	46%
2004	29%	18%	53%
2000	28%	23%	49%
1998	26%	28%	44%
1996	27%	40%	33%
1994	16%	31%	53%

Table 1: CHAOS Report for IT project performance from 1994-2008 (Standish Group 2009).

The original report in 1994 uncovered significant failure rates for IT projects (e.g. web application development, software development, systems integration) and estimated that these failures cost government agencies and corporations $80 billion - $145 billion per year (Standish Group 2009). The report features three categories of project performance – successful, challenged and failed. Successful projects refer to those that were delivered on time, on budget and with the desired scope. Challenged projects refer to those where the project was completed but was delivered late, over budget and without the required scope. Failed projects refer to those where the project was not completed, canceled or delivered something of no value to the customer.

While *The CHAOS Report* data suggests a marginal improvement in project performance between 1994 and 2008, it also highlights the continued challenges facing the IT industry and the project management discipline, as well as the recent dip in project performance between 2006 and 2008. Furthermore, the data is even more troubling when one considers that although 32 percent of projects were on time and on budget, most were originally over-estimated (Standish Group 2009). Focus groups indicated that “IT executives reported that they first get their best estimates, multiply by two and then add a half. It should not be surprising therefore, that the majority of these successful projects were already 150% over budget before they began” (Standish Group 2001).

In addition, the research suggests that the bigger the size of the project, which often times correlates to higher project complexity, the higher the failure rate (Standish Group 1999). Johnson states, "seventy-three percent of projects with labor cost of less than $750,000 succeed, but only 3 percent of projects a with labor cost of over $10 million succeed. I would venture to say the 3 percent that succeed succeeded because they overestimated their budget, not because they were managed properly" (COOK 2008). The research suggests there are three metrics that are key to predicting, with a high degree of certainty, whether or not a project will be successful – the project size, the project duration and the size of the project team. The research confirms what many project managers may have sensed all along; that smaller projects with shorter durations and smaller teams have a better chance at success compared to projects that are large with long durations. The 2001 report suggests a sweet spot for projects to be at four months, with four team members and a cost of less than $500,000 (Standish Group 2001).

Haas (2009) suggests that the reason for the overall improvement in projects, as seen in the CHAOS Report, is due to the following three factors: 1) Average project costs has been reduced by more than half since 2001; 2) more emphasis has been placed on using project managers and project management methodologies for complex IT projects; and 3) better methodologies and tools for change/configuration management have been developed and deployed. However, considering that the most recent data shows over two-thirds of all IT projects were over budget, over schedule or failed outright, and the fact that the Standish Group (2009) predicts that the number of critical projects will double every year it is apparent that the industry and the discipline still have a considerable challenge ahead to increase project performance to levels where management has confidence in the investments associated with these often large scale and dynamically complex projects.

Shenhar and Dvir (2007) recently conducted a study which spanned 15 years and more than 600 complex projects across the globe in business, non-profit and government sectors. Their research demonstrates yet another example of poor project performance when one considers that 85 percent of the projects studied failed to meet budget and schedule goals. On average, schedules were overrun by 70 percent and budgets by 60 percent (Shenhar Dvir, 2007). They concluded that "executives as well as project teams failed to appreciate up front the extent of uncertainty and complexity involved (or failed to communicate this extent to each other) and failed to adapt their management style to the situation" (Shenhar Dvir 2007). Furthermore, in the public sector, the Office of

Management and Budget (OMB) conducted a study in 2003 that concluded there was an unsustainable risk of failure among 771 projects with a total cost of $20.9 billion included in the fiscal year 2004 budget (OMB 2003).

b. Project Complexity and the Case for New Models

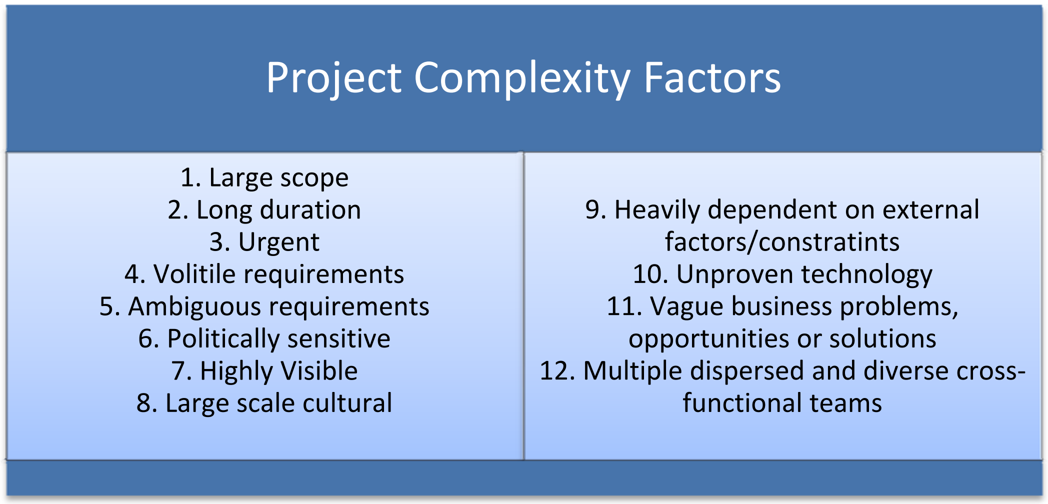
Complex projects require a team effort of experienced project managers, technical architects, developers, business analysts and executive leadership with a keen eye on strategy. These teams consist of complex adaptive social systems nested in diverse organizations, which in turn are nested in our increasingly dynamic economic, political and social environments. Once an organization defines the right projects to undertake, the question as to how to best do the projects (i.e. what methodology/process) becomes critical to the success of the project (Haas 2009). While traditional methods of project management are still relevant and useful, they are not always the best methodologies to select in today's environment. Projects in the 21st century require new methods of thinking and new approaches, some of which might replace traditional methods and some of which will augment them. The Project Complexity Model pioneered by Haas (2009) provides the project manager with a three step process to ensure that:

1. A project leader with the appropriate skill set is assigned
2. The project leader selects the appropriate project cycle
3. The management style selected is appropriate given the project's level of complexity.

The traditional approach to project management resembles a logical, linear, reductionist process to achieve a goal. It primarily involves the management of tasks and making sure those task are completed on time, within budget constraints and to the technical specifications of the customer. The methodology is a well defined one whereby the project manager/team performs the following linear progression:

1. Define the problem
2. Decompose the problem into its component parts (milestones, work-packages, activities)
3. Develop a plan to address the problem
4. Execute the work addressed in the plan
5. Integrate the decomposed parts into the final solution
6. Deliver/deploy the solution

The logic behind the traditional approach is that if we can reduce the problem down to manageable components, develop a plan to manage those components and then execute and control changes to that highly structured plan, we can therefore reduce the overall complexity and risk associated with the project (Project Management Institute PMI 2008, Gido and Clements 2009, Haas 2009, Lewis 2008). This methodology, born out of mathematical models and structured planning approaches to manage large aerospace engineering and construction projects of the 1950's, works well for projects where requirements are well understood and stable. However, projects in the information technology sector, and more specifically software projects, are often times highly complex, with volatile and ambiguous requirements; therefore, a new paradigm is emerging; one that is more complex, adaptive, agile and iterative (Loch et al 2006, Haas 2009).



Project Complexity Factors

1. Large scope
2. Long duration
3. Urgent
4. Volitile requirements
5. Ambiguous requirements
6. Politically sensitive
7. Highly Visible
8. Large scale cultural

9. Heavily dependent on external factors/constratints
10. Unproven technology
11. Vague business problems, opportunities or solutions
12. Multiple dispersed and diverse cross-functional teams

Figure 1: Twelve factors that produce complex projects (Haas 2009). Modern information technology projects often consist of several of these factors leading to increased project complexity and the associated increased project performance risks.

The Project Management Research Program is currently exploring the application of complexity theory to project management and what that area of science can provide to better understand the nature of complex projects (Haas 2009). In addition, new and more adaptive system development methodologies are emerging that emphasize agile and iterative techniques instead of the more traditional approaches, which place more emphasis on planning and control. A hybridization of these two methods is most likely the answer to handling different types of projects appropriately.

c. Software Development Projects

Software and web application development projects are complex in nature, involving technical knowledge-work that does not produce the linear relationships between system variables (e.g. the relationship between number of developers and lines of bug-fee code) often found in more tangible product production such as manufacturing and construction. For example, if one were to paint a house, the project’s schedule length would decrease almost linearly (up to a certain point) based on how many painters the project manager had for the task; this is not the case with IT projects and software/web application development in particular. Adding an extra developer to a project does not ensure that twice as many lines of bug-free code will be deployed. The complexity inherent in software development, and the project management practices that seek to control and improve its performance, have been well studied in recent decades and can be traced back as far as a report by the Comptroller General in 1979 citing the “software crisis” that existed in the federal government (Abdel-Hamid and Madnick 1991). The report concluded that “the government got for its money less than 2 percent of the total value of the contracts” (Abdel-Hamid and Madnick 1991, 3). Project performance for software development has improved marginally since then but is still deficient according to the Standish Group’s Chaos report (2009). Abdel-Hamid and Madnick (1991) produced a pioneering study in an attempt to understand the systemic issues involving software development and produce a working quantitative model that illustrates the complex interactions and causal connections between variables affecting software development.

Figure 2 illustrates a high level subsystem diagram for software development and demonstrates the key interactions between the planning, controlling, human resources management and software production components of any software development effort. Abdel-Hamid and Madnick (1991) suggest that in order to get a better understanding of the software development process we needed a fully integrated model that took other variables into account instead of the traditional practice of focusing solely on the software production subsystem.

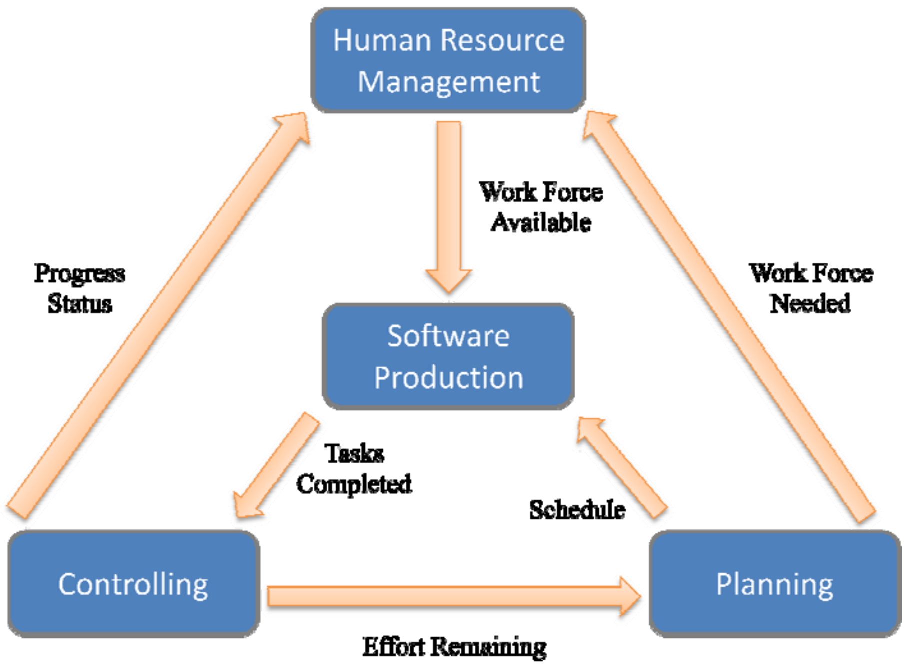


Figure 2: Fully integrated subsystem diagram for software development project dynamics (Abdel-Hamid and Madnick 1991).

The *Human Resource* subsystem is comprised of functions such as hiring, training and assimilation. Abdel-Hamid and Madnick (1991) recognized that these functions do not operate as independent and exogenous variables to the system, but rather as endogenous variables that both affect and are affected by the other subsystems. For example, the *Work Force Available* influences the allocation of human resources in the *Software Production* subsystem and both the *Controlling* subsystem and *Planning* subsystem have a direct affect on the *Human Resource* subsystem, and therefore the *Workforce Available (*Abdel-Hamid and Madnick 1991).

The *Planning* subsystem accounts for project estimation activities that directly affect both the project schedule and the work force needed to develop the software. Estimates are initially produced and subsequently revised throughout the life-cycle of the project. These estimates create the environment in which management interventions, such as adding more people to the project or adjusting the project schedule, take place and can lead to some of the unintended and counter-intuitive project behaviors referenced in Figure 13.

The *Software Production* subsystem comprises of four development activities: development, quality assurance, rework and testing. The Controlling subsystem represents those activities by which project managers track progress in development activities vs. the project plan and make adjustments. Unfortunately, tracking progress in software development is extremely difficult because the intended product (i.e. “software”) remains intangible for most of the development process. This makes it very difficult for both developers and the project managers to determine how far along the project has progressed (Abdel-Hamid and Madnick 1991, Hart 2009).

As previously mentioned, this is one of the major differences between information systems/software development project management control and accounts for much of the complexity and performance problems associated with managing software projects. The inability to accurately measure progress directly affects the project manager’s ability to effectively control the project schedule, budget and performance. Abdel-Hamid and Madnick (1991) suggest this is primarily the result of how software development projects are measured, using the surrogate variable, *consumptions of resources,* as a way to measure progress instead of a more tangible method you might see in manufacturing or construction. They assert that along with underestimation, this inability to precisely measure software project progress is one of the contributing factors in producing the 90% syndrome – a common form of project failure where estimates of the fraction of work completed reaches roughly 90% completion according to the original project schedule but then stalls as the task completion rate starts to decrease. In some cases the project takes just as long if not longer to finish the final 10% as it did to finish the first 90% (Abdel-Hamid 1990, Abdel-Hamid and Madnick 1988, 1991, Ford and Sterman 2003, Lyneis and Ford 2007, Hart 2009).

d. The 90% Syndrome in Systems/Software Development

In the project management and software development literature the 90% syndrome refers to a type of project failure where estimates of the fraction of work completed reaches roughly 90% according to the original project schedule but then stalls (Abdel-Hamid 1988). Sterman (1992) concisely defines the syndrome by stating that “a project is thought to be 90% complete for half the total time required” (1992, 1). In more layman’s terms, projects have an uncanny way of appearing to be on target and proceeding as planned until they near their end; however, even though project management planning and control took place, the rate of progress stalls as the scope of the endeavor grows, the resource requirements increase and/or the errors made early on in the development process lead to schedule pressure, schedule slippage, overtime, poor product quality and potentially yet more rework.

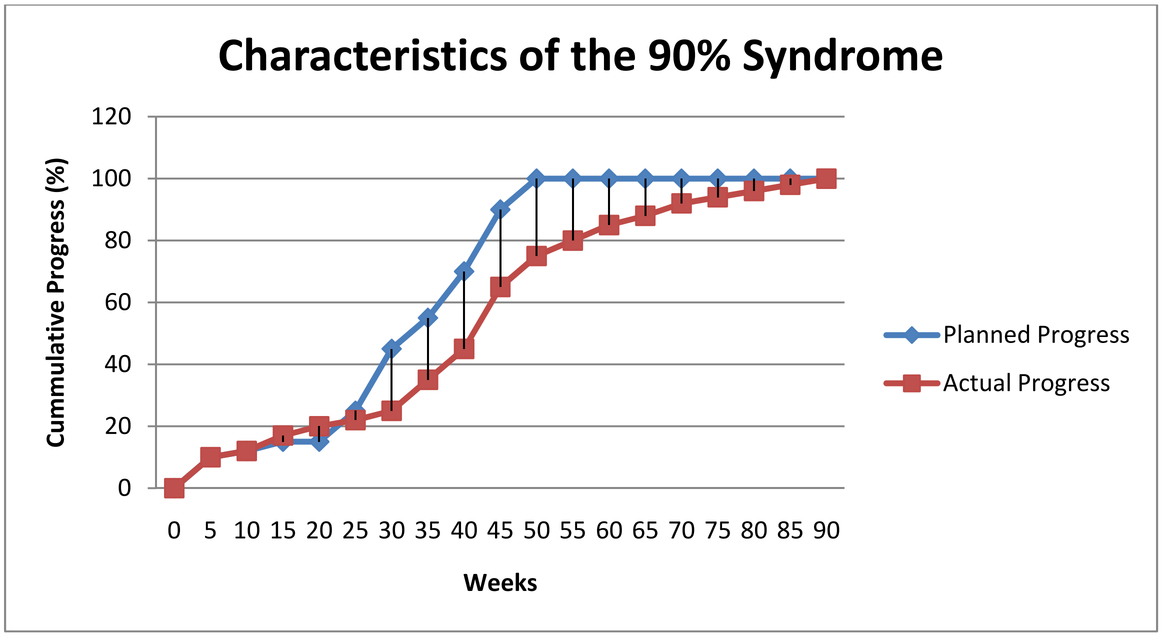


Figure 3: Classic S-shaped limits to growth representation of the 90% syndrome. Cumulative progress increases at a decreasing rate once the project nears 90% completion according to the original project plan (Ford and Sterman 2003).

There are several reasons cited in the literature for this project phenomenon. Abdel-Hamid and Madnick (1991) attribute it primarily to the interaction between two factors: project size (i.e. scope) underestimation and man-day underestimation (i.e. human resources). As one might expect, if the project scope is underestimated from the beginning there is a greater chance for the 90% syndrome to appear due to the following characteristics surrounding many software development projects. Early on in the project, the ability of the project manager to effectively measure progress is hampered because software is mostly intangible for much of its development and progress is often measured by the rate of resource expenditures instead of actual accomplishments (e.g. bug-free lines of code or functioning software modules); this in turn creates the illusion that the project is on schedule (Abdel-Hamid 1991).

Once the project progresses into its later stages, the discrepancy between resources expenditures and completed tasks becomes increasingly apparent to both the project team members and the project manager. This produces a better appreciation for the actual amount of work left; often times the inevitable discovery and underestimation of rework exacerbates the syndrome, leading to a prolonged stall or outright failure (Ford and Sterman 2003). Furthermore, progress can continue to decline as a result of ill-advised managerial responses to the schedule failure. For example, such “corrective actions” as adding more developers, pushing the deadline, engaging in concurrent development and/or requiring overtime, produce what Ford and Lyneis (2007) categorize as “ripple” and “knock-on” effects. These actions often activate feedback dynamics, induced and exacerbated by management interventions, which can have the unintended consequence of producing an increase in the error rate, further reduced productivity and further delays.

The second factor Abdel-Hamid and Madnick (1991) attribute to the syndrome is man-day underestimation. According to their model, the syndrome is more severe when man-day requirements are underestimated than it is when size is underestimated (Abdel-Hamid and Madnick, 1991). At first this might appear to be somewhat counterintuitive; however, a closer examination of the case study and model results reveals that when size is underestimated the syndrome is less severe for two reasons: 1) because problems with the estimation of project scope tend to be detected faster in the project life cycle than problems with labor estimates and size underestimation, and 2) because when size is underestimated, and subsequently detected in the early stages of the project, man-days estimates are often revised appropriately (Abdel-Hamid and Madnick, 1991).

Man-day requirements often remain undetected until late in the life cycle when the majority of the budgeted man-days have been consumed; this occurs because it isn't until the later stages of a project when the budgeted man-days are exhausted and the team members are able to effectively perceive how productive they have actually been (Abdel-Hamid and Madnick 1991). Bringing additional resources into the project during its critical late stages (when the rework cycle starts to dominate), instead of earlier in the project lifecycle, increases the project's potential for Brooks' Law (i.e. adding resources to a late software projects only makes it later), as well as other supporting ripple and knock-on effects (Lyneis and Ford 2007), further degrading schedule/project performance (Abdel-Hamid and Madnick 1991).

In addition to Abdel-Hamid and Madnick's assertions on the causes for the 90% syndrome, Ford and Sterman (2003) also provided insights through their study of concurrent engineering projects. Concurrent engineering refers to a method by which certain tasks, work packages and/or project milestones are developed in parallel rather than serial formation. For example, in the case of software development, a project manager might choose to overlap (i.e. work in parallel) the design and development tasks in an effort to reduce the total schedule length of those task, as compared to working them in serial order.

Ford and Sterman (2003) developed a system dynamics model to better understand the interactions between the process structure of concurrent development and the project team members' behavioral decision-making processes. Their model demonstrates the strength behind the system dynamics methodology; its ability to integrate managerial decision-making into the various physical information processes involved in any software development effort. The results from their research illustrate how the interaction between development processes (e.g. overlapping activities/task sequencing, activity durations and rework), as well as the behaviors of management and developers deliberately concealing rework (i.e. “the Liars Club”), creates a detrimental dynamic leading to unplanned iterations and a lower quality product at a higher than estimated cost (Ford and Sterman 2003). They assert that concurrent engineering actually increases the risk of producing the 90% syndrome because it increases a project’s vulnerability to multiple iterations and errors, as well as actually increasing the fraction of work requiring changes or additional iterations (Ford and Sterman 2003).

Their research outlines the “Liar’s Club” as a social/behavioral phenomenon in projects whereby project teams conceal rework in order to avoid the responsibility of failure, prevent blame escalation or retaliation from peers and to solve their own problems under cover and free from management intervention. This behavior illustrates the importance of considering (i.e. factoring them into models) individual and team behaviors, a practiced widely used in system dynamics modeling, when devising policies to improve project performance in concurrent development projects (Ford and Sterman 2003). According to Ford and Sterman (2003, 217), “Process changes cannot improve concurrent development project performance if they do not also address the behaviors that drive iteration cycles such as the policy of concealing rework requirements.” In other words, without considering and addressing how people truly behave in socio-technical systems, the project manager can continue in vain to tweak development processes ad infinitum, only to realize that the project team’s socio-cultural incentive structure continues to wreak havoc on the final project outcome.

The literature referenced in this research focuses on the use of system dynamics modeling and simulation, which provides a better understanding of the project behaviors, such as the aforementioned 90% syndrome, as well as quantitative data to support the anecdotal evidence encountered by many project managers in the field. The goal for project managers should be to first develop a richer understanding of dynamic project behaviors, which include individual, management and project planning/estimation behaviors. Armed with that knowledge, they must use it to develop better strategies for handling project perturbations in a way that mitigates the risk of the 90% syndrome.

These strategies should include, but are not limited to the use of, 1) better estimation tools for both project size and man-days requirements, remembering that project estimates create project behaviors and vice versa; the methods to also properly account for rework cycles and their critical position late in the project life-cycle, as well as the man-day considerations needed to effectively handle them, 2) improved methods to account for, manage and adjust the socio-cultural incentive structures associated with the concealment of rework (i.e. the Liar’s Club); the literature suggests agile development as a methodology to increase cycle speed and mitigate the syndrome’s risks; however, management’s communication styles/patterns, as well as their policies towards rework, must not institutionalize the desire to conceal problems and hide the “bad news,” and lastly, 3) the development, and use of, commercially available software project management simulation tools (e.g. system dynamics models) that are widely accessible, intuitive, extensible, relevant and provide project managers with a robust learning environment from which to learn and become better dynamically driven decision makers.

Each one of these suggested strategies represents a component that must be considered in any attempt to improve the marginal success rate of software development and information technology projects. Simply addressing whichever component solution is conveniently located in close proximity has the potential to deliver minor incremental improvements, no improvements, or worse yet, aggravate the development process and deliver worse results than before.

III. Augmenting Traditional Project Management Methodologies using System Dynamics Lessons Learned

a. What is System Dynamics?

System Dynamics is a modeling and simulation methodology pioneered by Jay Forester at the MIT School of Management in the 1950's and grounded in the mathematical theories of nonlinear dynamics and feedback control. Initially termed Industrial Dynamics, it was used by management teams who recognized the counterintuitive nature of their production systems and needed a problem solving method for analyzing the complex and dynamic nature of their production processes, inventories and services (Sterman 2000). Sterman (2000) concisely, yet precisely, describes it as, "a method to enhance learning in complex systems." Since its inception, the methodology has been applied across a wide spectrum to solve problems and assist managers in the policy and decision making process.

The discipline is predicated on the following main premises (Sterman 2000, Meadows 2008, Senge 1990)

1. The behavior of any system, in other words its output, is principally caused by its structure and associated interrelationships and interdependencies between system components or variables.
2. The system structure includes not only the physical aspects of the system but the policies and procedures that govern organizational decision-making.

3. The whole is greater than the sum of its parts. Our intuitive judgment proves to be unreliable in understanding system behavior over time even when we think we fully understand the individual components of the system.
4. All systems involve feedback processes which often produce counter-intuitive system behavior.
5. The key to understanding a system is to understand the causal connections and feedback structures that exist and how those interrelationships dynamically influence system behavior over time.

One of the most fertile application areas for the system dynamics methodology is within the discipline of project management and, more specifically, the discipline's application to software development and construction (Sterman 2000). In order to properly understand system dynamics, and its many contributions to project management, it is important to grasp some of the core concepts central to systems thinking, such as event-oriented vs. systems-oriented worldviews, mental models, single-loop vs. double-loop learning, stocks, flows, feedback and delays.

i. Event-oriented vs. Systems-oriented world views

Central to the system dynamics methodology are the concepts of a systems-oriented world view **(**figure 4), as opposed to the event-oriented worldview **(**figure 5). Quite often, these system characteristics govern the dynamic behavior of the systems themselves, leading many systems thinkers to the realization that a system's output is more a function of the relationships between system components, than simply the sum of each component. In other words, the output for a system/project is much more than the sum of all its parts (Sterman 2000, Meadows 2008, Senge 1990).

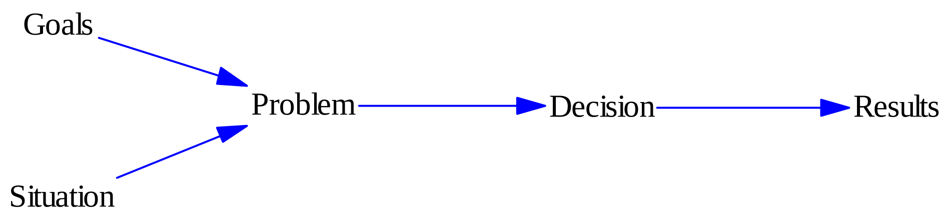


Figure 4: An event-oriented world view assumes the problem is static in nature and develops as a result of some event or combination of events (Sterman 2000).

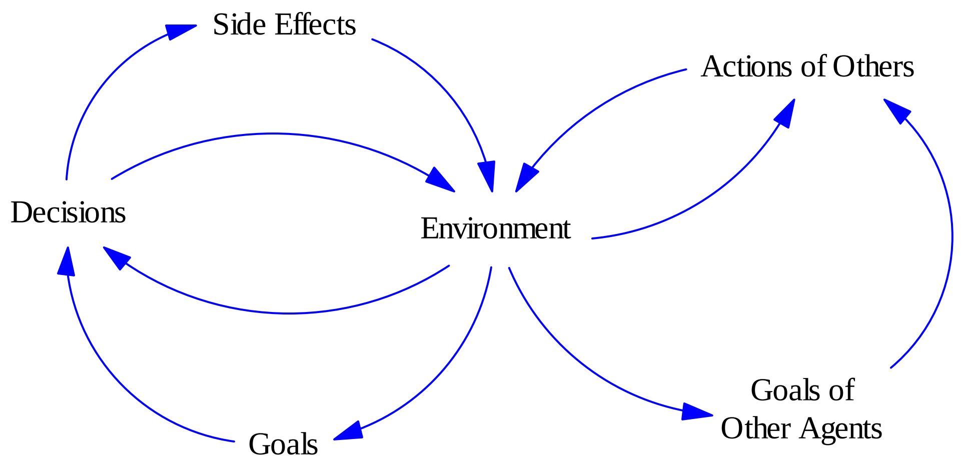
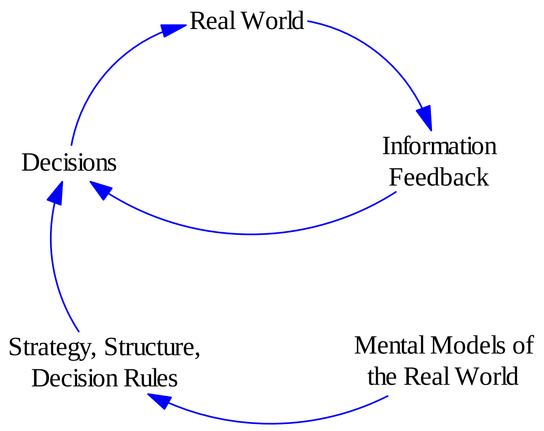


Figure 5: A systems-oriented world view assumes the problem is dynamic in nature and arises as a natural consequence of dynamic relationships between the various system components. It broadens the scope of the model boundary, accounts for other actors, accounts for feedback dynamics and attempts to account for potential unintended consequences (Sterman 2000).

For example, a project manager would be hard pressed to understand, or predict, the true outcome of any project by simply dissecting it down to its component part tasks, estimating, planning and scheduling the project. That approach is ignorant to the more holistic systems approach, which realizes that the slightest perturbations in any task can have cascading and escalating effects throughout the lifecycle of the project, leading to increases in errors/rework, costs and schedule overruns. This higher level strategic perspective is considered to be one of system dynamics' major contributions to the discipline of project management.

ii. Mental Models and Single vs. Double-Loop Learning

Another core concept in system dynamics is the notion of mental models, which refer to a person's preconceived understanding about how some particular aspect of the world works (Sterman 2000). Senge (1990) describes mental models as "deeply ingrained assumptions, generalizations, or even pictures of images that influence how we understand the world and how we take action." When we fail to recognize or subject our mental models to scrutiny or testing, our ability to learn and better manage complex systems is minimized. A systems approach to project management provides the project manager with a methodology to formally identify, test and adjust mental models so he/she can effectively understand and analyze all of the data presented instead of embarking on a path using a static, faulty or biased cognitive foundation. Constantly challenging our mental models leads to the process of double-loop learning (Sterman 2000).


Real World
Information Feedback
Decisions
Strategy, Structure, Decision Rules
Mental Models of the Real World

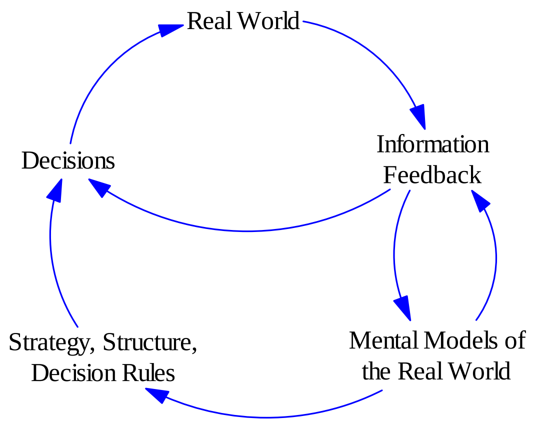

Real World
Information Feedback
Decisions
Strategy, Structure, Decision Rules
Mental Models of the Real World

Figure 6: Mental models are important variables in problem solving activities and should be perpetually challenged as information feedback increases. These diagrams illustrate the difference between single loop learning (left) and double loop learning (right); they demonstrate how double loop learning uses information feedback from the system to refine, or in some cases replace, deeply held biases associated with the object of study, which in turn go on to affect our strategies and policies and produce yet more information feedback (Sterman 2000).

Galileo and Lord Kelvin both emphasized the importance of studying and quantifying the phenomena we wish to understand, with Kelvin stating, “to measure is to know.” A common maxim in the project management community also states that you cannot manage what you cannot measure, which is also commonly referred to as “management by fact,” compared to the opposite philosophy of intuitive decision-making. Management by fact is advantageous compared to intuitive decision-making because it exposes the mangers mental models and challenges them to be continually re-examined when the data supporting those mental models changes (i.e. double-loop learning). This is of course is not without difficulty when one considers the myriad of variables present in any decision-making process that are either un-quantifiable, or very difficult to establish consensus on how to properly quantify the variable.

For example, in software development, variables such as customer satisfaction, team moral, team knowledge/expertise, project scope and latent software errors are all representative of variables Hart (2008) refers to as “fuzzy” or “soft variables;” these variables challenge the manager to provide data, and as Hart (2008) suggests, eventually “encounter the wall of measurability.” Thus, as managers, if we accept the philosophy inherent in managing by fact we are forced to ignore important soft variables present in our project. Ignoring these variables can produce negative systemic effects when intervention strategies are planned without regard to the effect these soft variables have on the intervention’s outcome; sometimes producing counter-intuitive side effects that ultimately degrade project performance.

Properly integrating these soft variables into operational models is difficult practice; however, taking the easy way out and excluding them can lead to poorly performing intervention strategies and what Sterman (2000) and Meadows (2008) refer to as policy resistance, or “the tendency for interventions to be delayed, diluted, or defeated by the response of the system to the intervention itself” (Sterman 2000, 5). As Senge (1990, 60) wisely states, “the easy way out usually leads right back in.” System dynamics for project management provides a methodology for effectively building both theoretical and quantitative models that integrate hard and soft variables into one model, properly illustrating the cause-effect relationships and systemic feedback structures.

iii. Stocks, Flows and Causal Loop Diagrams

If one were to conceptualize the previously mentioned concepts as the philosophical foundations for systems thinking, then the concepts of stocks, flows and causal loop diagrams would be considered its language. Peter Senge (1990) says the following regarding causal loop diagramming and the language of systems thinking:

> "Language shapes perception. What we see depends on what we are prepared to see. Western languages, with their subject-verb-object structure, are biased toward a linear view. If we want to see system-wide interrelationships, we need a language of interrelationships, a language made up of circles. … Such a language is important in facing dynamically complex issues and strategic choices, especially when individuals, teams and organizations need to see beyond events and into the forces that shape change."

More than any other tool, stock and flow diagrams lie at the foundation for systems thinking. Such diagrams describe dynamic system structure, and can be used to provide insight on the behaviors those structures produce. Without causal loop diagramming and the stock and flow models they seed, there is limited potential to expand system dynamics' application via advanced tools such as: computer modeling and simulation, and digital learning environments that mimic flight simulators.

Stocks are the system state variables, sometimes also referred to as system "reservoirs" because they represent accumulations in any system (e.g number of software defects, number of developers on a project). Meadows (2008) describes them as elements in a system that you can count feel or measure at any given point in time. They are accumulations and, therefore, are the main source of system buffer and delays. Flows on the other hand are rates (i.e. processes) that affect the stock accumulations in an increasing or decreasing fashion. If the sum of all inflows is greater than the sum of all outflows the stock level will increase; if the sum of all outflows is larger the stock will decrease, and if inflows and outflows are equal, the system state remains in dynamic equilibrium (Sterman 2000, Meadows 2008).

Understanding how system stocks and flows work, and their associated feedback structure, is critical to understanding the true nature and output of any system. For example, figure 7 depicts a simple population model stock and flow structure in an attempt to illustrate this important concept with a very simple, narrow boundary system (Sterman 2000). The *Population* stock is the system state variable we are concerned with understanding. It represents a snapshot in time and can be quantified using people as the units. The stock has two associated flows. On the left, there is the *Births* flow that adds people to the population stock. The left side represents a self-reinforcing (i.e. positive) feedback loop structure (R1), which indicates that as more people are born, and the population grows in size, there will be more births, which in turn increases the population. This kind of self-reinforcing feedback represents the classic exponential growth archetype (Senge 1990, Sterman 2000, Meadows 2008). To the right of the population stock is the *Deaths* flow. This side represents the balancing (i.e. negative) feedback loop structure (B1), which indicates that as the population increases, the number of deaths per unit of time will also increase, which in turn then decreases the population stock in a “goal seeking” fashion.

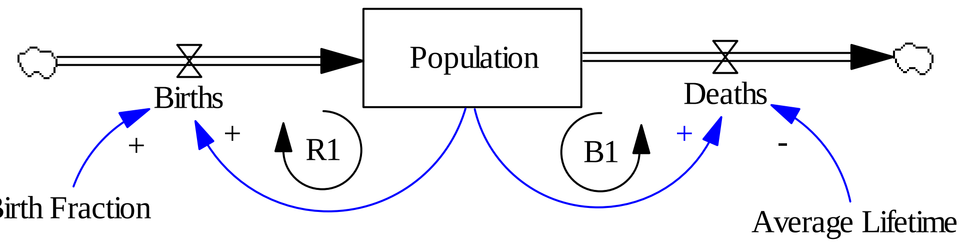


Figure 7: Population Model Stock and Flow Structure (Sterman 2000).

The different polarities associated with each feedback arrow indicate how one variable influences the other. In the case of *Births,* for example, as either the *Birth Fraction* variable or the *Population* stock increases, so will the *Births* flow increase. The positive polarity sign indicates that these variables move in the same direction. However, on the right side balancing loop, as the *Average Lifetime* increases the number of deaths decreases. The negative polarity indicates that these variables move in opposite directions.

iv. Feedback and Delays

The feedback structure of any system, and the delays associated with that system, predominantly account for the system's complex, dynamic and sometimes counterintuitive behavior. Simply put, feedback refers to the interactions among system variables. A feedback loop is formed when the inflows and outflows of a particular stock (e.g. population) are affected by changes in that stock. So in the case of figure 7, the size of the population stock directly affects both *Births* and *Deaths* flows and creates a feedback loop for each. These interactions comprise the structure of the system and, therefore, demonstrate how systems cause their own behaviors. Moreover, different systems with similar feedback structures (i.e. system archetypes) tend to exhibit similar dynamic behaviors (Senge 1990, Sterman 2000, Meadows 2008). Figure 7 demonstrates both types of feedback loops – positive (R1 self-reinforcing) and negative (B1 balancing). Positive feedback loops are runaway processes that produce exponential growth (e.g. compounding interest of bank account). Conversely, balancing feedback loops provide system stability and illustrate goal-seeking behavior (e.g. an increase in inventory increases sales which then decreases inventory).

System delays, in addition to feedback structure, significantly determine system behaviors. Sterman (2000) describes delays as “a process whose output lags behind its input in some fashion.” Delays often produce stock oscillations, especially in a balancing feedback loops. In IT projects, and more specifically software/web application development, delays can affect the project’s outcome and complicate our ability to understand and potentially change the system/project behavior. Hart (2008) suggests there are three ways in which delays impact a system. First, delays produce reactions that are not necessarily proportional to the system’s observed conditions. Second, project team members act as if their decision making processes are based on current (real-time) data when in fact systemic delays assure this is not the case. And lastly, delays affect our ability to learn and properly decipher the cause-effect relationships among system variables. In software development, delays can be witnessed in the lag-time inherent between a customer’s perception of software quality and its actual quality, the delay associated with new-hire productivity gains and from a traditional project management perspective, the reality that project status reports often times do not truly reflect the project’s progress and are not always based on current information regarding the project (Hart 2008).

b. Systemic Project Dynamics in Software Development – The Dynamics of Schedule Pressure

The occurrence of the aforementioned 90% syndrome in projects is well documented in the literature (Abdel-Hamid 1990, Abdel-Hamid and Madnick 1988, 1991, Sterman 1992, Ford and Sterman 2003, Lyneis and Ford 2007, Hart 2009) and the causes for the syndrome are extensive and complex in nature. While Adbel-Hamid and Madnick (1991) assert that the main causes can be attributed to underestimation and the imprecise measurement of project progress, other studies by Ford and Sterman (2003), Lyneis and Ford (2007) and Hart (2008) suggest the causes can be much wider and involve other factors such as the interaction between factors such as “the rework cycle” and project staffing trends throughout the lifecycle of the project.

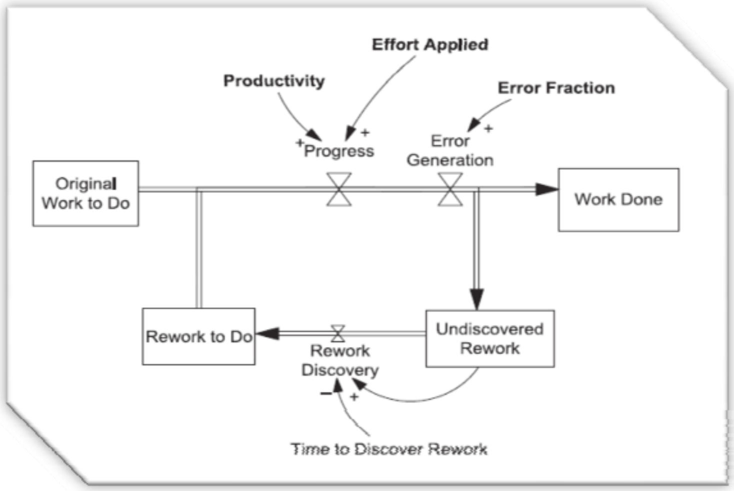


Figure 8: A typical development project rework cycle. Work packages move from the original work to do stock to the work done stock. In the process errors are discovered and sent back through the development process for "rework" (Lyneis and Ford 2007).

The various behaviors, interrelationships and causal connections that typify most software and web development projects are illustrated in Figures 9, 10, 11, 12 and 13. These dynamics represent common behaviors referenced in the project management literature and are important learning tools for project managers. They provide a foundation for shared understanding of typical behaviors that can lead to poor project performance, or outright failure. Furthermore, they can be used by project team members to identify project pitfalls, communicate systemic issues, design and adjust project policies and perform elementary risk-analysis.

Figure 9 illustrates the core theoretical sub-model to the comprehensive model found in figure 13. It integrates a few of the more common policy decisions found in information technology projects and centers on the S*chedule Pressure* variable, which when increased or decreased throughout the project lifecycle can become both cause and effect for second the third order project effects . Lyneis and Ford (2007) refer to these as “ripple” and “knock-on” effects, which are the key causes of policy resistance.

Various project control efforts (e.g. adding more developers, working overtime) cause ripple effects which are commonly understood as side effects to these well-intentioned managerial interventions. These ripple effects then produce second and third order effects (e.g. errors/rework, out of sequence work), amplifying problems in the project and feeding back throughout the systems structure to produce counter-intuitive systemic project behaviors. Using these conceptual causal loop diagram models to effectively communicate these dynamics is an important initial step towards management’s improved understanding of the complex dynamics at play and their ability to learn and develop appropriate intervention strategies (Hart 2008).

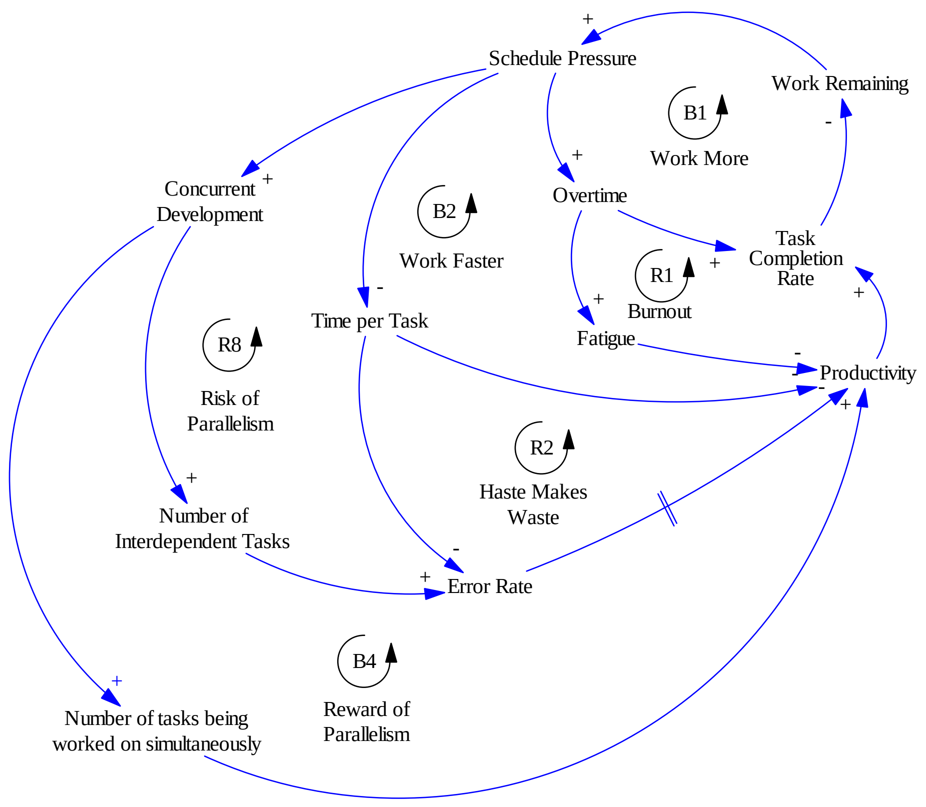


Figure 9: Theoretical model illustrating common project schedule policy decisions and their systemic cause-effect relationships. Adapted from Sterman (2000).

A software/web application development project under *Schedule Pressure* can exhibit various behaviors that either lead to a reduction in *Schedule Pressure* or unintentionally feedback to produce an unintended increase in *Schedule Pressure*. When faced with Schedule Pressure, a project manager has an array of options to choose from should intervention become necessary. For example, overtime can be instituted to make up for lost time, the amount of time dedicated to each work package or task can be reduced (i.e. Cutting Corners, B2 and R2 from figure 9), or the project manager can decide to engage in concurrent development as referenced in both the R8 and B4 loops. Each one of these interventions is designed to reduce schedule pressure by increasing the rate of development; however, they each can produce unintended side effects that work to counter those intended gains in productivity (Sterman 2000). The B1 loop in figure 9 illustrates how an increase in *overtime* can increase the *task completion rate*, decrease the *work remaining* and form a feedback balancing loop by reducing the *schedule pressure*; however, R1 illustrates that an increase in *overtime* can also lead to developer *fatigue*, a decrease in *productivity*, and a decrease in the *task completion rate*, which then leads to an increase in the amount of *work remaining* and ultimately leads to more *schedule pressure* (Sterman 2000).

Alternatively, the project manager could opt to reduce *time per task* (R2 loop in figure 9), which would be designed to boost *productivity* and in the end reduce *schedule pressure*; however, speeding up development by cutting corners can also increase the *error rate* because developers become less concerned with following sound systems development methodologies and instead focus on task completion regardless of the negative costs associated with it such as software bugs, errors and reduced functionality. Therefore, regardless of the intent of the policy, the reduction of *time per task* can often times lead to reductions in *productivity* and subsequently feed back to ultimately increase *schedule pressure* and perpetuate the negative feedback loop (Sterman 2000).

Another option the project manager might consider is to engage in concurrent development (R8 and B4 of figure 9 by overlapping project work packages and/or phases in attempt to increase the development rate and boost productivity. This is an increasingly common practice in systems development (Ford and Sterman 2003) and is not without its own set of negative repercussions. Ford and Sterman (2003) assert that regardless of some success, implementing concurrent development has proven to be challenging for many project managers mainly because it increases the amount of communication overhead (i.e. information transfers) between the project phases/tasks; therefore, more tasks are begun using preliminary or incomplete information and specifications. This leads to an increase in the number and length of iteration cycles, sometimes producing what Taylor and Ford (2006) refer to as “tipping points” in development projects whereby tipping point feedback structures push the project into “firefight mode” as errors pile up in ever increasingly taxed rework cycle and, therefore, further degrade project performance.

B4 of figure 9 illustrates how an increase in *concurrent development* strategies increases the number of *tasks being worked on simultaneously*, which leads to an increase in *productivity* and then feeds back to reduce *schedule pressure*. However, the increase in concurrence can also increase the *interdependency between tasks*, which can increase the *error rate* and after some delay lead to a decrease in *productivity* as more and more errors enter the rework cycle (Ford and Sterman 2003, Taylor and Ford 2006).

c. Brooks’ Law

As the task completion rate of a project begins to decrease management interventions such hiring and firing rates can increase, leading to the classic counter-intuitive phenomenon found in software and systems development known as Brooks’ Law (figure 10), which states that adding developers to a late software project only makes it later (Brooks 1975).

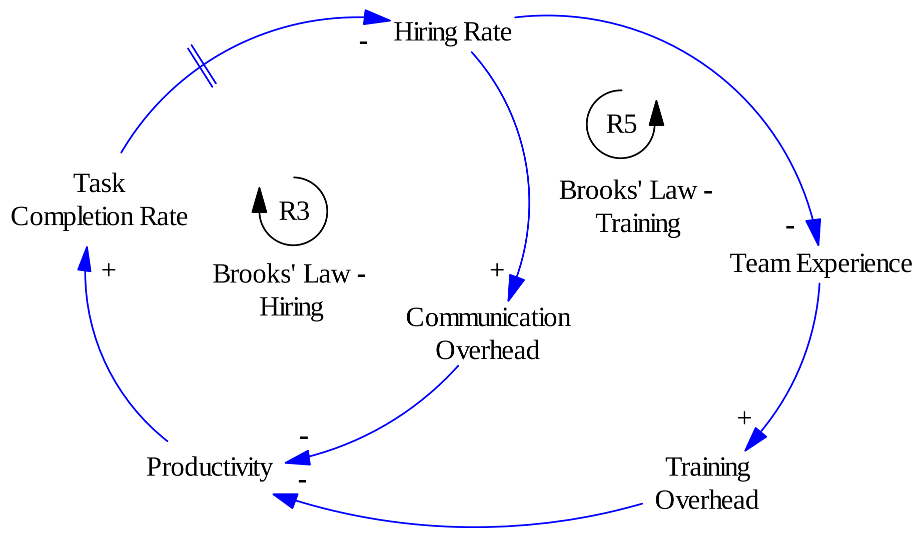


Figure 10: Brooks Law. Adding developers to a late software project can make it later due to the increase in communication and training required. This increase leads to lowered productivity, lowered progress and the potential for more increases in hiring developers to work on the delayed project. The balancing feedback to this loop is not represented here but will be covered in more detail in figure 13.

As the *task completion rate* begins to decrease, project managers might decide to intervene by adding more developers (i.e. an increase in the hiring rate), which increases the *communication overhead* for the project, lowers the *productivity* and ultimately feeds back to further degrade the *task completion rat*e (Abdel-Hamid 1988, Abdel-Hamid 1990, Lyneis and Ford 2007). An increase in the hiring rate can also dilute the *experience levels* for the team (R5 in figure 10), which in turn increases the *training overhead*, lowers the *productivity* and ultimately feeds back to further decrease the *task completion rate*. Furthermore, as team experience erodes it is also common to see an increase in the *error rate* which also feeds back to lower *productivity* and pave the way for potential increases in both *schedule pressure* and *hiring rates*.

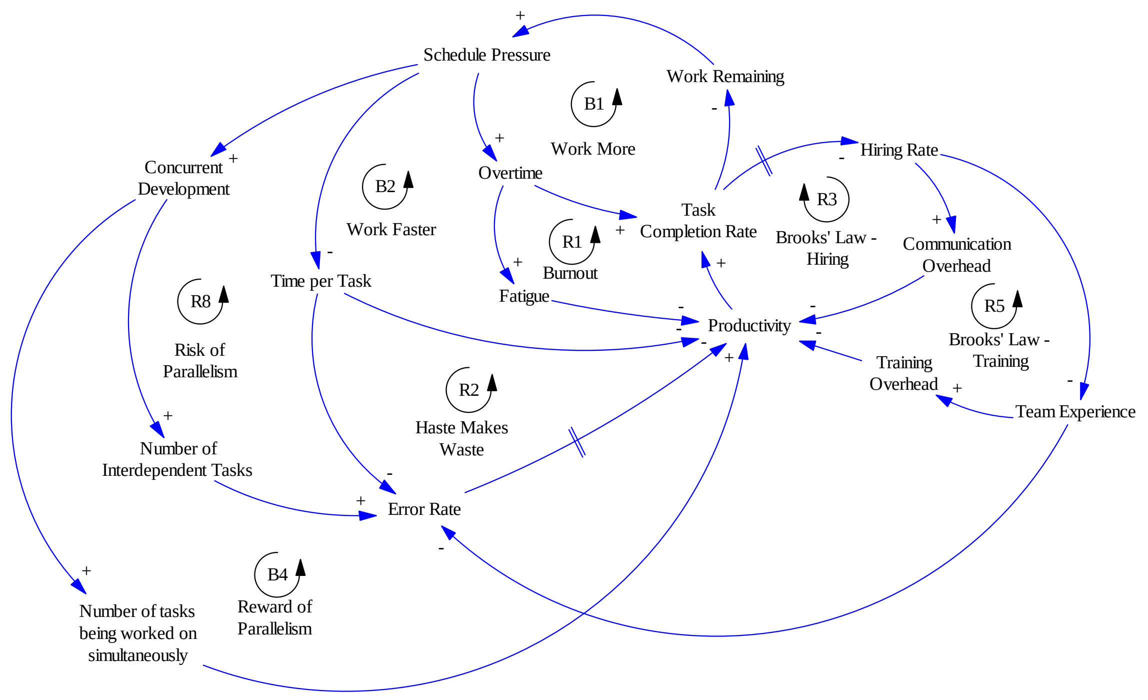


Figure 11: Causal loop diagram showing the interactions and relationships from figure 9 after integrating the effects of Brooks' Law (figure 10) on project productivity.

d. Parkinson's Law

Having tried all the other options, a project manager might also decide to push the deadline of the project in an attempt to provide more time for the project team and increase the probability for success (B3 in figure 12). While this practice increases the time remaining and therefore should result in a decrease in *schedule pressure*, it also can lead to Parkinson's Law which states that work expands to equal the time allotted for it (Abdel-Hamid and Madnick 1989). A deadline push that increases the time remaining on the project also increases *slacking off* which lowers *productivity* and feeds back to further increase *schedule pressure*. In addition, it can also lead to a common software and systems development practice called *gold plating*, whereby developers, with more time due to the deadline push, decide to add superfluous features to software versions even though they were not scheduled to be part of the planned release cycle (Lyneis and Ford 2007). While this practice might initially appear to be a good use of the extra time, it often times leads to additional errors that otherwise would not be in the rework cycle.

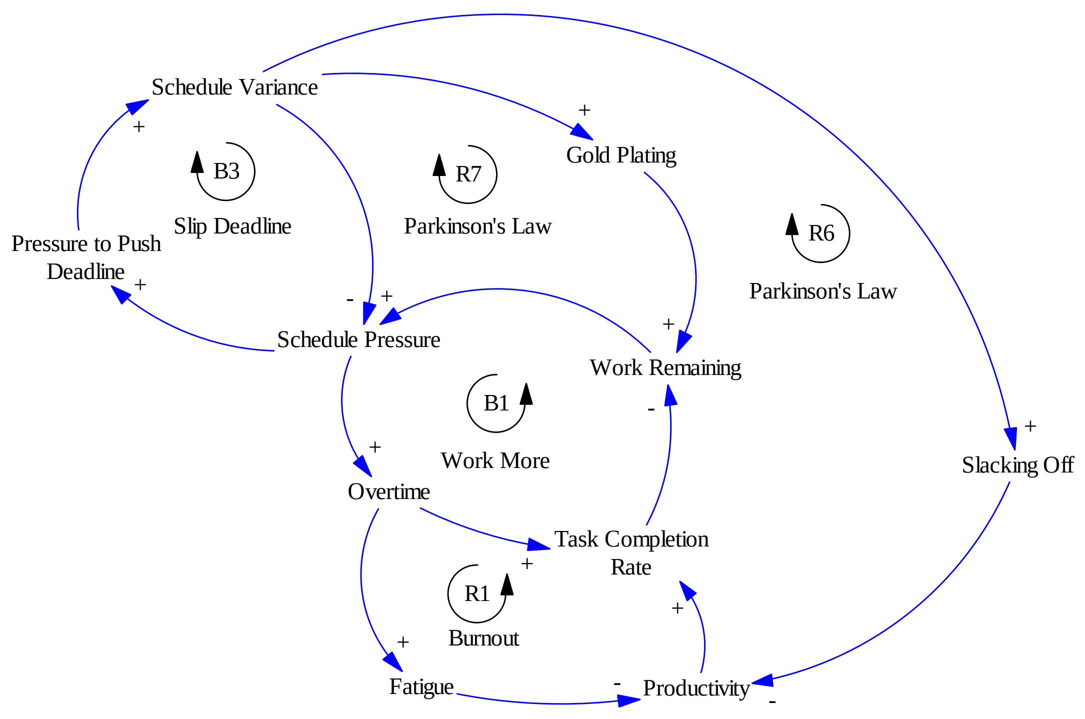


Figure 12: Causal loop diagram showing the integration of variables such as "deadline slips" and Parkinson's Law and the effects of those variables on the project team's productivity.

e. Integrated Project Dynamics Model

Once we integrate figures 9-12 into figure 13 the complex dynamics associated with projects becomes increasingly apparent and the policy decisions made by project managers are exposed as both cause and effect of both positive and negative feedback, as well as both increases and decreases in project productivity. The model illustrates the difficulty associated with project policy decisions and the counter-intuitive nature of those policy effects as they interact with an increasing amount of system variables, which in turn alter system states in a non-linear fashion.

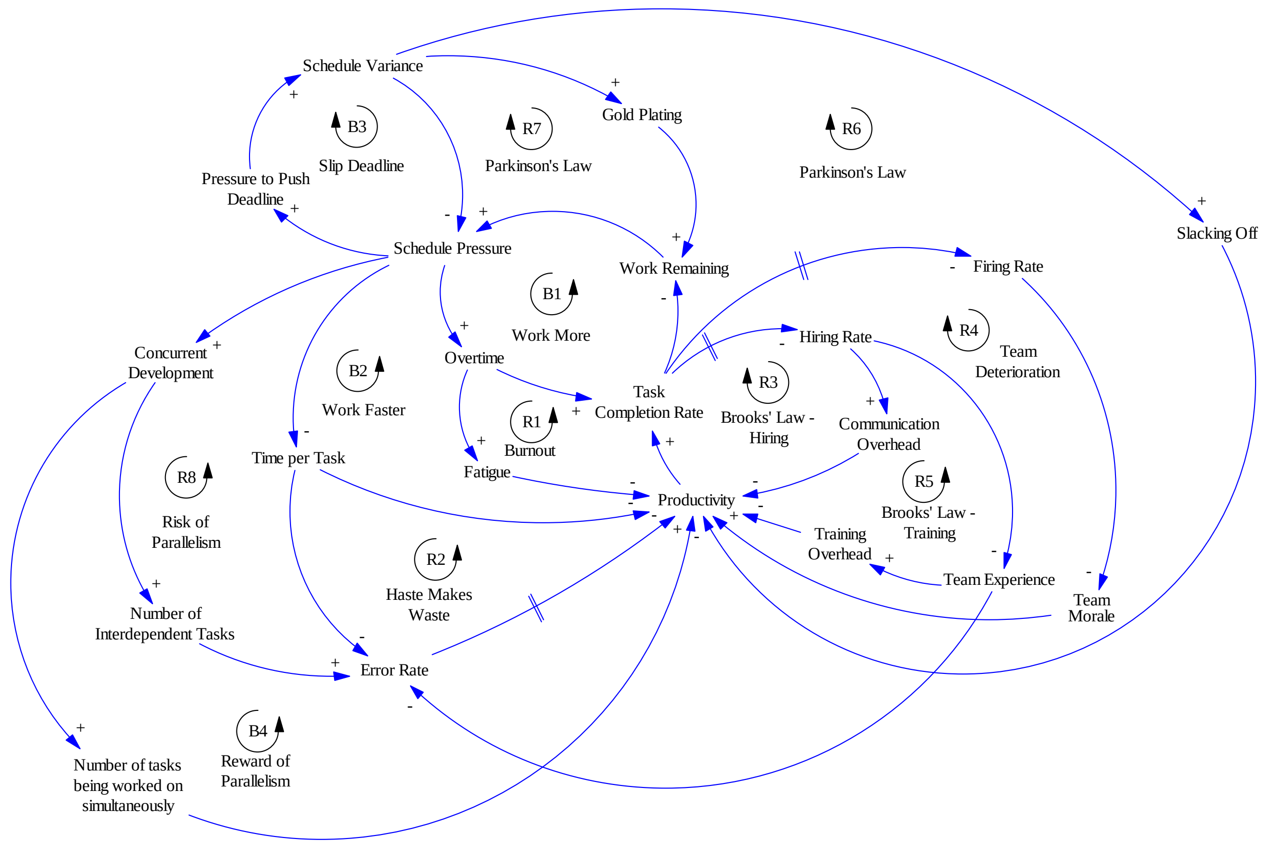


Figure 13: Conceptual model designed to illustrate typical systemic project behaviors and their causal connections Adapted from Sterman (2000), Lyneis and Ford (2007), Ford and Sterman (2003).

The conceptual model referenced in figure 13 does not account for some additional variables that could contribute to schedule pressure such as, aggressive deadlines set by management, overly optimistic assumptions of productivity and/or quality, customer interventions, satisfaction levels or increases in customer feature requests. For the purposes of this study those variables will be considered outside the boundary scope and therefore exogenous to our system. The relevant concept is that *Schedule Pressure*, the central variable for this model, is common in projects, and management interventions, in the form of traditional project control techniques, represent both *Schedule Pressure* cause and effect.

f. Client Interactions

It is important to note that while the majority of research referenced in this paper refers to ripple and knock-on feedback structures associated with internal project actions, policy decisions and interventions, further reinforced by figure 13's model boundary which implicitly considers client interaction as exogenous to the system, external sources, such as customers or clients, can trigger or amplify some of the aforementioned project dynamics referenced in figure 13 (Rodrigues 1998). Rodrigues (1998) states that client initiated scope creep, low-balled project proposals followed by change orders, as well as deadline slips, all contribute to feedback dynamics that directly affect and degrade project performance.

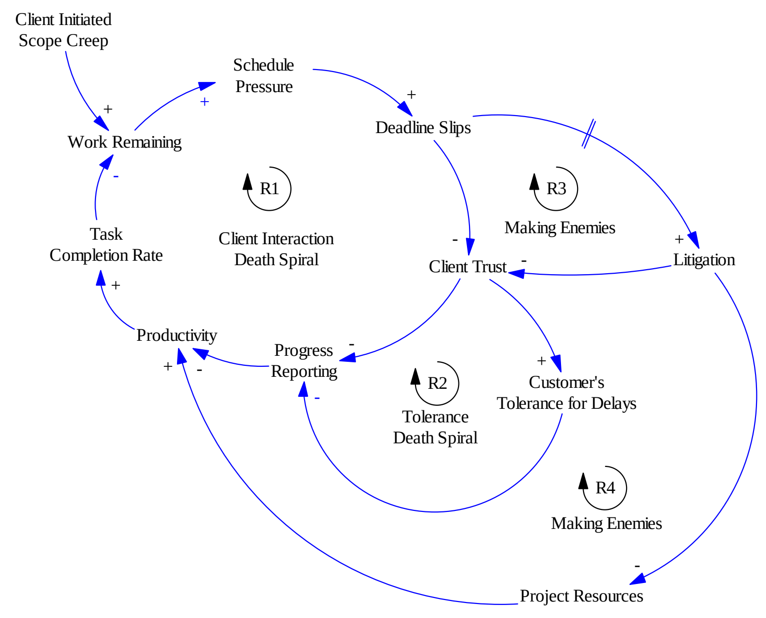


Figure 14: Causal loop diagram illustrating the feedback structure associated with external influences (e.g. clients/customers) on the internal project dynamics referenced in figure 13.

Figure 14 illustrates some of these external source dynamics and demonstrates a few important areas of concern regarding the impact of client behavior on project performance. The dynamics associated with figure 14 could also be integrated into figure 13 and would demonstrate the added effect of *client initiated scope creep* on *work remaining*. *Client initiated scope creep* can increase work remaining on a project as more and more change requests or feature adds pile up. This can lead to an increase in *schedule pressure*, *deadline slips* and a decrease in *client trust*. As *client trust* begins to degrade, *progress reporting* usually begins to increase, which reduces *productivity* and further degrades *schedule performance*. As *deadline slips* mount and *client trust* degrades, their *tolerance* for future deadline slips also decreases, which in turn produces more *schedule pressure*, more *progress reporting* and feeding back to further degrade *schedule performance*. In the worst case contracting scenario, significant *deadline slips* can ultimately lead to *litigation* efforts initiated by the client (USN, Laverghetta and Brown 1999, Sterman 2000), which divert and/or decrease *project resources*, leading to a reduction in *productivity* and a further decrease in *schedule performance*.

g. Project Estimation Techniques

Given the importance of the project deadline and the schedule pressure that results from that deadline, it is not surprising that some of the literature (Abdel-Hamid and Madnick 1991) suggests how projects begin, and more specifically how their scope and associated schedules are estimated, is of critical importance to the outcome of the project. They assert that the most common method is one referred to as “estimation by analogy,” whereby the project team uses historical data and experience as the guiding factors to produce future work package estimates (Abdel-Hamid and Madnick 1991). Abdel-

Hamid and Madnick (1991) suggest that this common tendency to base future estimates of a project’s (or work package’s) cost and schedule based on similar past project reveals organizational bias in the scheduling process and can ultimately lead to higher project costs and increasingly longer project schedules over time.

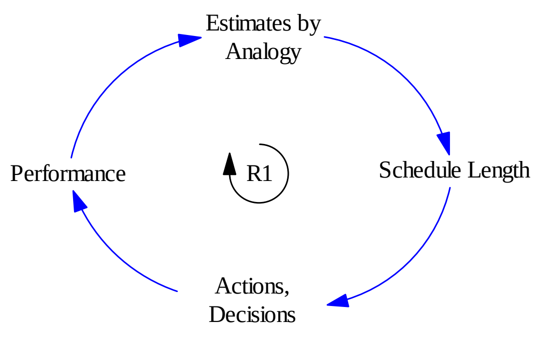


Figure 15: Estimation by Analogy. Estimates influence schedules which influence actions and decisions by the project team and management, which affects project performance ultimately affecting the next project's estimates.

The logic follows that if one work package is analogous to another from previous experience it should therefore require the same amount of time to complete. This method makes several dangerous assumptions, most of which, if not all, can turn out to be false as the project progresses and significantly reduce the probability of project success. First, it assumes that the historical data driving the new estimate is not itself already bloated with deliberate and/or excessive slack. Second, it assumes the same people, or different people with the same skills sets, will be working those tasks. In addition, if there are different people involved with the same skill sets it also assumes that those people will work, make decisions and deliver an identical result in fit, form and function as compared to the people that produce the original work package from which the estimate was made. Third, it assumes that the interdependency of project tasks are not themselves dynamic and that their relationships may vary in linearity from project to project. Lastly, it assumes that management interventions, actions or decisions, as well as those involving the project team, will be the same for the new work package as they were for the one the new estimate was based on.

Abdel-Hamid and Madnick (1991) assert that management's tendency to cling to unrealistic schedules by adding developers, instead of adjusting the project deadline to more appropriately reflect the level of effort required to finish the project, is political in nature. They suggest this happens for two reasons; first, project managers are hesitant to show a slip in the project schedule "too early" in the project, and second, if they re-estimate now they risk doing it again later and looking bad twice (Abdel-Hamid and Madnick (1991). This type of political decision making in projects can also become unintentionally institutionalized and consequently ignored by management as one of the potential causes for reductions in project performance, as well as unnecessary increases in *schedule pressure* and the down-stream dynamics associated with that increase.

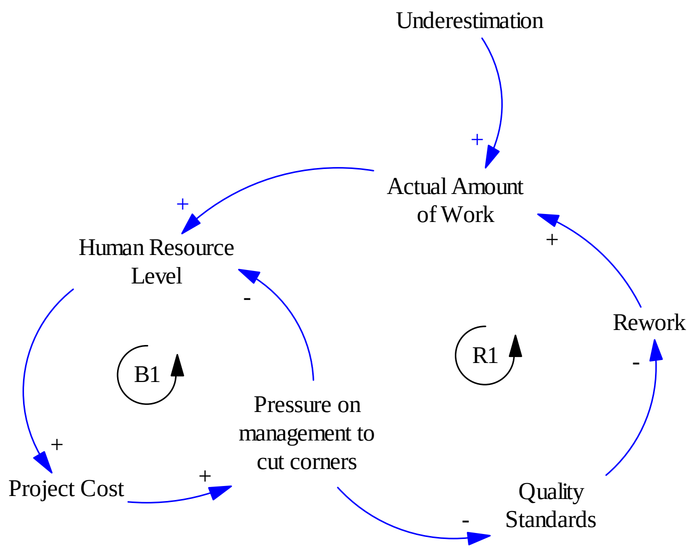


Figure 16: This causal loop diagram illustrates management's tendency to cling to unrealistic project schedules (underestimated) by simply adding more people to the project. This produces an increase

in project cost leading to an increase in pressure to cut corners (B1). That pressure can reduce human resource levels, but does so at the expense of quality standards. As quality standards decrease rework increases, further increasing the project's actual amount of work.

Some of this political decision making is guided by conventional wisdom in the project management community, as well as research from two empirical studies and an informal review of over 500 completed contracts, stating that when "a contract is more than 15 percent complete, the [final] overrun at completion will not be less than the overrun to date, and the [final] percent overrun at completion will be greater than the percent overrun to date" (Christensen 1993, 45). In addition, Christensen (1993, 45) states that "once a contract is 20 percent complete, the cumulative Cost Performance Index (CPI) does not change more than 10 percent; in fact, in most cases it only worsens." Given these project management industry rules of thumb, it becomes obvious that the pressure to not show a schedule slip early in the project life cycle produces a dynamic whereby project managers choose to add developers and cost to the project instead of analyzing estimates and potentially re-estimating scopes and schedules to achieve a better target end date, which in turn reinforces the dynamics associated with Brooks' Law (figure 10).

1V. System Archetypes as Project and Organizational Learning Tools

The study of system archetypes as project learning tools is another contribution the system dynamics methodology makes to the project management discipline. These generic system structures provide a conceptual foundation so that project teams can identify common generic project variables and their associated dynamics, communicate their cause-effect relationships, map them to the project or issue at hand and plan effective intervention strategies for mitigating problematic project behavior. Effective learning usually requires some level of pattern recognition; system archetypes provide these project patterns, which can be analyzed to improve performance.

Both Meadows (2008) and Sterman (2000) provide detail on system archetypes as they relate to generic system behavior; however, Hart (2008) takes these system archetype concepts a step further using software engineering, and various real-world software case studies, to illustrate archetypes as they relate to the software development process. Hart's (2008) research, and associated exploration into system archetypes applied to software engineering, provides an IT project manager with a solid foundation for analyzing typical software project behaviors and designing intervention strategies to mitigate the risks inherent in those software engineering system archetypes. While there are several system archetypes to learn from, the scope of this paper will limit us to a discussion on two areas common to systems/software development – software quality and process improvement efforts.

a. Drifting Goals – Software Quality Erosion

The first archetype we will explore is the Drifting Goals archetype. Senge (1990), Meadows (2008) and Hart (2008) all describe this archetype as a situation whereby organizational/project goals are allowed to drift or slowly erode to a state that is ultimately unacceptable to the organization; however, because it happens slowly over an extended period of time it goes unnoticed or becomes institutionalized into the business process. Meadows (2008, 122) refers to this phenomenon as the "boiled frog syndrome" and suggests, "some systems not only resist policy and stay in a bad state, they keep getting worse." A frog's biology for sensing danger is hard-wired to sudden environmental changes, not gradual ones; therefore, if you place a frog in water at room temperature and slowly turn up the heat to a boiling level it will be content until it is too late to save itself (Senge 1990). Project delays, communication issues and inaccurate project control data can all have the same effect on a project, turning a slow gradual decline in performance into outright project failure.

The systemic structure of the drifting goals archetype, as referenced in figure 17, provides the project manager with a generic understanding of the dynamics surrounding goal erosion behavior. Figure 18, takes this concept a step further and applies it to software development to illustrate the unintended consequences of goal erosion with regards to software quality standards.

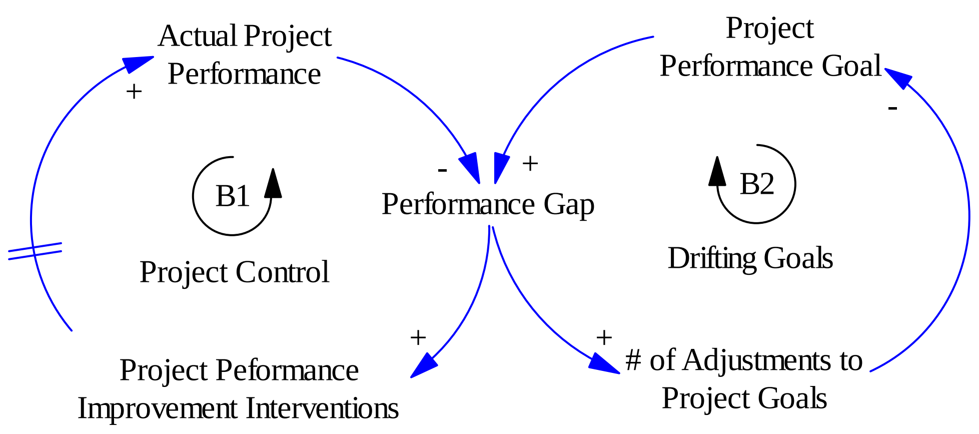

Actual Project Performance
Project Performance Goal
+
-
+
-
B1
Performance Gap
B2
Project Control
Drifting Goals
+
+
Project Peformance Improvement Interventions
# of Adjustments to Project Goals

Figure 17: Generic structure of the drifting goals archetype

The behavior over time we are concerned with here is either the increase or decrease of both *Project Performance Goal* and *Actual Project Performance* over time. The *Performance Gap* variable is the difference between the *Project Performance Goal* and the *Actual Project Performance*. When the gap increases, it produces two systemic reactions; the first (B1) is the project control loop which states that as the *Performance Gap* increases, so do project team interventions to improve performance, and as Project Performance Improvement Interventions increase Actual Project Performance increases, albeit most often after some amount of delay. This delay is important because as it increases it produces a sentiment among team members that fixes are failing and/or improving the project performance is a fruitless effort, which ultimately triggers the introduction of the B2 loop – the Drifting Goals loop. This feedback mechanism results in a balancing feedback loop that increases the *# of Adjustments to Project Goals* when the *Performance Gap* increases. Those interventions then reduce the *Project Performance Goal* and ultimately reduce the *Performance Gap*, which makes the team believe they are succeeding, even though the reality is that should this behavior continue it can result in a continued degradation of project performance when one accepts a variable like software quality as an appropriate measure of project performance.

Figure 18 illustrates these drifting goals dynamic as it applies the erosion of software quality over time. For the purposes of this example, we will continue the practice of considering software quality as an appropriate measure of project performance, which according to Hart (2008) is a function of delivered functionality and software defects.

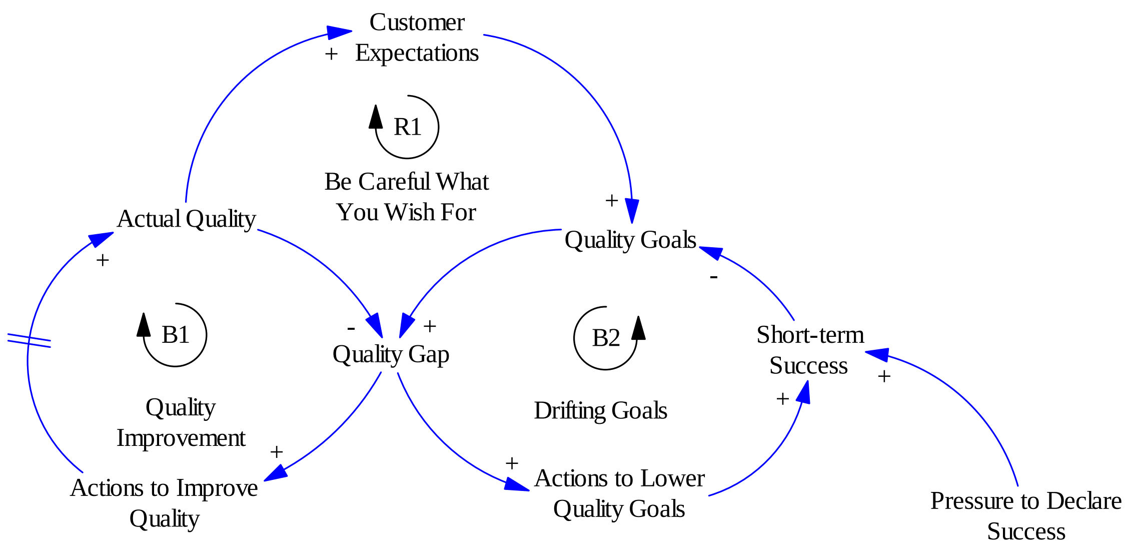


Figure 18: The erosion of software quality. Software project related example of the drifting goals archetype (Hart 2008).

Most, if not all, software development projects are forced to deal with the cost vs. schedule vs. quality dilemma. Software quality can degrade over time as quality goals shift due to increased pressure to deliver products on time and under budget (i.e. *pressure to declare success*). Figure 18 is designed to illustrate the dynamics behind this dilemma. Like figure 17, the Quality Gap is the difference between the *Quality Goal* and the *Actual Quality*. Hart (2008) suggests that this gap produces three main dynamics. First are the system interventions or *Actions to Improve Quality* (B1), such as testing and process improvement techniques. Second are the *Actions to Lower Quality Goals* such as releasing a product prior to testing it properly and reworking errors. These actions are usually a result of *pressure to declare success* and the fact that the *Actions to Improve Quality* are often expensive and time consuming; the logic is that the customer is still using the product right now and not complaining that much so why not. Hart (2008) suggests that in general, the reality is that people tend to tolerate software that is beneath their expectations or standards. Third are the *Customer Expectations*, which when considered as part of the system equation produce the potential dynamic whereby software of acceptable quality gets worse over time. If the *Actual Quality* of the software decreases, so does the *Customer's Expectation* of that software, which reduces the Quality Goals, reduces the Quality Gap, reduces the *Actions to Improve Quality* and in turn further reduces the *Actual Quality* of the software. The erosion of software quality, as illustrated in figure 18, is therefore a systemic problem including both the project team and the customer.

b. Limits to Growth – The Paradox of Process Improvements

The second archetype to briefly explore is the limits to growth archetype mentioned in the literature (Senge 1990, Meadows 2008, Hart 2009). Hart (2008) suggest this to be the generic structure associated with the aforementioned 90% syndrome in projects; and while Lyneis and Ford (2007), Ford and Sterman (2003) and Abdel-Hamid (1991) don't specifically reference the limits to growth system archetype in their research, their research on the 90% syndrome describes a system structure and behavior that is consistent with this archetype.

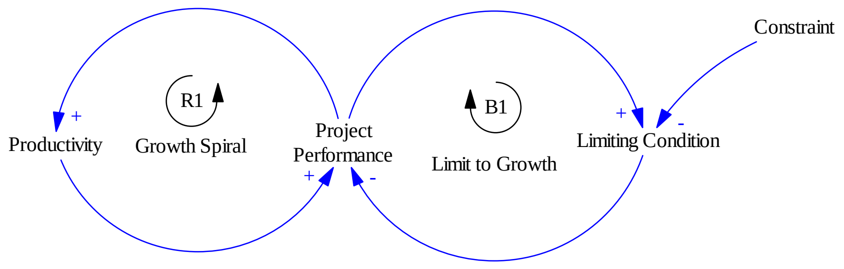


Figure 19: Simple generic system structure of a limits to growth archetype.

This limits to growth archetype is simple in nature and revolves around the *Project Performance* variable, which when increased causes a positive feedback loop (R1), increasing *Productivity* and further increasing Project Performance. This growth spiral does have its limits though, represented in the B1 loop, whereby some constraint on the system produces a limiting condition (e.g. the number of developers writing code is a constraint that produces a limiting condition), which ultimately balances out the system, reduces project performance and ultimately demonstrates the limits to growth archetype.

Figure 20 illustrates this limits to growth concept as it applies to project management with an example Hart (2008) refers to as the process improvement paradox. For the purpose of this example, the causal loop diagram has been kept relatively simplistic so as to clearly demonstrate the archetype; however, this dynamic could easily be integrated into figure 13's overarching structure, and if you look closely, while the variables names are different, the concepts overlap with two of the loops – B1 and R1.

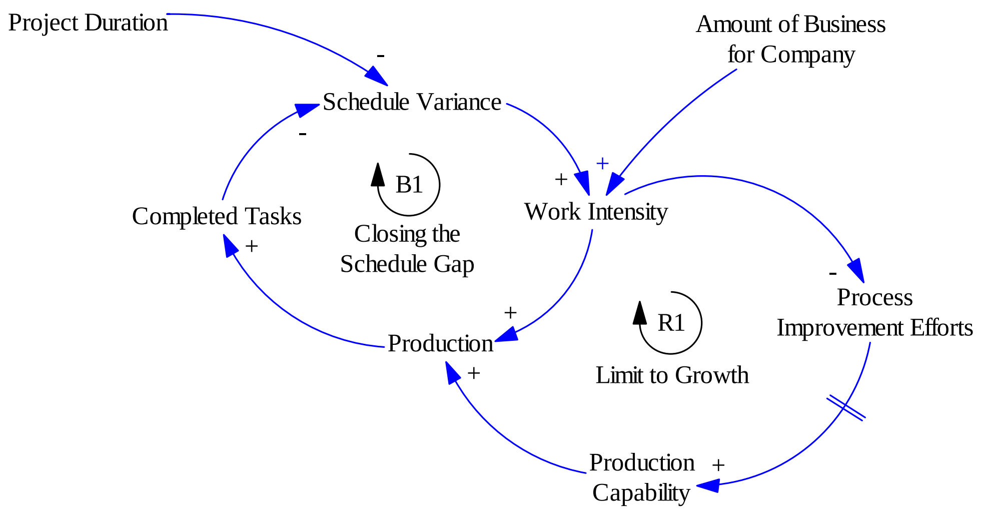


Figure 20: Project management specific example of a limits to growth archetype (Hart 2008).

B1 of figure 20, or the *Closing the Schedule Gap* balancing loop, illustrates that as *Schedule Variance* (i.e. the difference between where you hoped to be in the project schedule and where you actually are) increases, *Work Intensity* increases, which can cause an increase in *Production*, an increase in *Completed Tasks*, and therefore, a decrease in *Schedule Variance*. However, increases in *Work Intensity* can also lead to a degradation of *Process Improvement Efforts* among the project team, as more time is dedicated to completing tasks and less time is therefore dedicated to improving project/production processes in order to increase *Production Capacity*. R1, and more specifically the relationship between Work Intensity and Process Improvement Efforts, clearly demonstrates the limits to growth archetype in systems and provides project managers with the foundation for applying these concepts, as well as their associated interventions, to better improve their project performance.

c. Inter vs. Intra Project Learning

The organization, or team learning process, is of critical importance for project success and the study and application of system archetypes aid in that learning process. Kotnour (2000) suggests there are two forms of learning in a project environment – “inter-project learning” and “intra-project learning”. The goal of inter-project learning is to increase project and team learning acumen; it is the knowledge assimilation, combination and dissemination of lessons learned across projects to facilitate knowledge creation, and ultimately, better project performance (Kotnour 2000). The dissemination process is critical for organizations interested in mitigating the risks of perpetuating poor project performance across both similar and disparate projects. Assimilation and combination involves the post-mortem analysis of a project, which is an important project plan phase (Kotnour 2000).

However, this inter-project learning (i.e. post-mortem) process alone does not provide a panacea for solving problematic project behaviors and mitigating the risks of their associated outcomes. Equally important is to engage in the accompanying practice Kotnour (2000) describes as “intra-project learning,” whereby the project team develops methods and processes for capturing near real-time project data (e.g. number of software bugs released to production) and integrates that data into the project lifecycle, creating knowledge and improving project decision making. While this practice should accompany inter-project learning, it often times does not, as more effort and resources are often times applied to the project’s close out phase, or post-mortem analysis, in support of the next project.

The study and development of system archetypes primarily falls within the domain of inter-project learning; however, as this paper will discuss later, they can also be used for intra-project learning when one considers the theoretical integration of system dynamics models with the traditional project management process framework as outlined by the Project Management Institute's Project Management Body of Knowledge (2008). Further discussion regarding this novel practice will be presented later in Chapter V.

Intra-project learning focuses on the task of individual projects with the goal of producing a successful project outcome, to build individual and team capabilities, and ultimately, to support inter-project learning as the data that is captured during the project is then assimilated, combined and distributed in support of that mission (Kotnour 2000). It identifies, analyzes and solves problems during the course of a project as opposed to the post-mortem methodology, a method widely used in both traditional project management processes and system dynamics analysis to projects (USN, T. Laverghetta and Brown USN 1999), which relies on project data captured during a project to provide insight into how to improve performance on future projects.

Unfortunately, because the very definition of a project according to the Project Management Institute's Body of Knowledge (2008, 5) is that of a "temporary endeavor undertaken to create a unique product, service or result," it stands to reason that solely relying on inter-project learning as a strategy for increased project and organizational performance is myopic; particularly when one considers that projects vary in requirements, complexity and dynamics from one project to the next. This consideration underpins the foundation for research, (Rodrigues and Bowers 1996, Rodrigues and Williams 1998, Williams 1999, Williams 2003, Taylor and Ford 2006, Lyneis and Ford

2007) in both the system dynamics and project management communities, that attempts to improve traditional project management processes and methods by modifying, replacing, and in some cases, augmenting them to better govern contemporary project dynamics and effectively integrate new models into the existing project management framework as outlined by PMI (2008).

V. Discussion: Integration of System Dynamics with Traditional Methods and Suggestions for Future Research

Williams (1999) and Rodrigues (1997) provide a rough outline for this new methodology – the integration of dynamic models with traditional project management methodologies, and other areas in the literature (Lyneis and Ford 2007, Taylor and Ford 2006, Lyneis and Cooper 2001, Rodrigues and Bowers 1996, Klein 2007, Hart 2008, Haas 2009) also suggest the need to re-examine the use of our traditional techniques for project management in an attempt to develop better strategic, as well as tactical, methodologies. Many of the current operational models, while useful, are lacking in a holistic/systems approach to project management; therefore, integration of system dynamics models into the entire project management framework, and not solely as a post-mortem analysis tool to support inter-project learning and knowledge dissemination, represents a novel practice in the project management field and one that could potentially benefit project teams as well as the organizations they support.

This augmented project management framework would be designed to better manage projects in real-time and provide functioning quantitative models that project managers could use during a project instead of just after the project in support of the next. These models would be integrated into the traditional project management toolset (PERT, CPA,WBS, OBS, Network Diagramming) and lifecycle in a way that shares project data (e.g. % of tasks completed for a particular work package, or number of software defects released to production) in near real-time with the previously mentioned operational tools used to plan and control projects.

At this stage in its evolutionary development, system dynamics models applied to project management are predominantly used as post-mortem tools and occasionally as models used prospectively (e.g. estimation techniques). Progress towards this next step of integration into the entire project life cycle (see figure 21) is currently more of an academic exercise than a mature methodology. The literature at this point does not suggest any real-world case of this novel approach to augmenting the project manager's tool set. The literature does, however, consistently emphasize the need for new models to adapt to the ever increasing complex nature of information technology projects, as well as to improve the mediocre performance that continues to plague the industry.

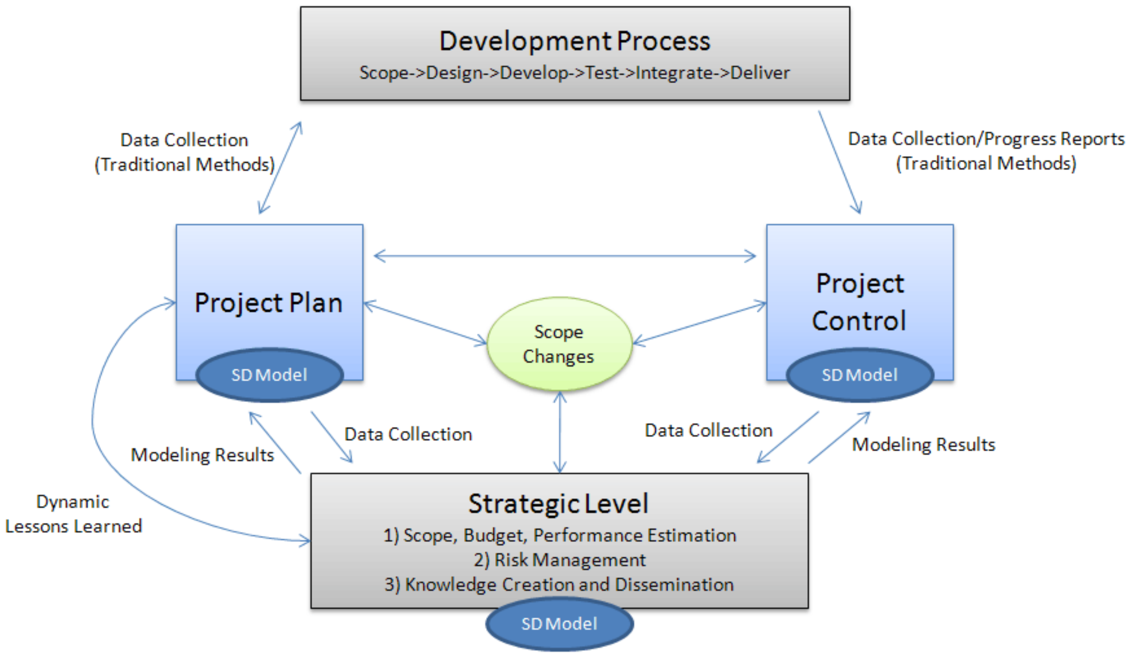

Development Process
Scope->Design->Develop->Test->Integrate->Deliver
Data Collection (Traditional Methods)
Data Collection/Progress Reports (Traditional Methods)
Project Plan
SD Model
Scope Changes
Project Control
SD Model
Modeling Results
Data Collection
Data Collection
Modeling Results
Dynamic Lessons Learned
Strategic Level
1) Scope, Budget, Performance Estimation
2) Risk Management
3) Knowledge Creation and Dissemination
SD Model

Figure 21: Conceptual model of the integration of system dynamics models (SD Models) in the traditional project management framework. Lessons learned and modeling 'what if" scenario results are aggregated at the strategic level and continuously fed into the traditional plan and control project management framework.

Models that originated from the defense and aerospace engineering projects of the 1950's are useful but need to be augmented with more contemporary, quantitative and dynamic models that allow project managers the ability to test "what-if" scenarios (e.g.

will adding developers at the later stages of this project make the project stall even more) before, during and after their projects in an attempt to facilitate knowledge creation throughout the entire project life-cycle and organizational learning prowess among all project team members. Properly and continually calibrated system dynamics models, using as close to real-time project status data as possible, integrated into a project manager's tool set (see figure 21) could serve many purposes including:

1. Effective project management training, mentoring and skill set building. The use of CLDs and quantitative stock and flow simulation models could provide project managers with a learning environment that enables them to see the feedback structures present in a particular project management problem (e.g. client scope changes) and run simulations to test/validate their managerial intuition on how best to intervene. These tests could serve to educate managers on the 2nd and 3rd order effects of their interventions and where in the system it is best to intervene (i.e. which levers to adjust).
2. Improved project estimation and risk analysis/mitigation techniques
3. Dynamically adaptive project plans/network diagrams
4. Scenario analysis used to plan intervention strategies should problems arise throughout the course of the project and post-mortem analysis to support the project's close-out phase and facilitate inter-project learning. Intervention strategies must also be adaptive given the complex nature of modern-day IT projects and the challenges this type of knowledge-work presents.

Rodrigues and Williams (1998) developed a conceptual model called the Project Management Integrated Model (PMIM) for how this integration might look and broke it down into both a System Dynamics Operational Model (SDOM), as well as a System Dynamics Strategic Model (SDSM). They suggest using the integration of system dynamics models as a way to develop a more sophisticated approach to project network models that would include feedback processes designed to deliver more effective modeling of project activity durations and costs (Rodrigues and Williams 1998). In addition to the integration of system dynamics models with the established project management processes, Rodrigues (2001) also suggests the use of system dynamics modeling within the Project Management Body of Knowledge risk management processes, thereby providing an alternative, yet complementary, method for managing project risks.

Unfortunately, at this stage in its evolution, the literature suggests little has been done to normalize, implement and test these integrated models in practical organizational situations. While this represents undoubtedly a massive undertaking, it is one which has the potential to positively influence the direction of project management research and practice; further research in this area is warranted. Simply identifying lessons learned at the end of a project, and using those lessons as a means to improve future project performance, is insufficient to advancing the performance rate of information technology projects and the project management discipline in general. The delay associated with team learning and the development of effective management interventions must be reduced; knowledge creation, combination, assimilation and dissemination, as well as risk mitigation strategies and tactics, must be developed and implemented throughout the entire life-cycle of the project, not just at the end of a project and in support of the next.

In addition to the integration of system dynamics models and simulation into the traditional project management framework, the use of system archetypes as a project management learning tools should also be further researched, developed and applied to specific areas of project management such as information systems projects and web application development. While there will always be overlap in the concepts explored in these system archetypes, different project types, such as the development of complex and diversely distributed cloud computing applications, call for specific examples to be developed, validated and circulated amongst project management professionals. This knowledge creation and dissemination process could facilitate the development of a new, more systems-thinking oriented world view for the discipline.

However, while the use of system dynamics as a project team learning tool is of particular importance to advancing the project management field, we should not limit the discussion to simply using these tools as strictly an education tool. More research must be done to propagate the use of system dynamics as not just an academic exercise, but also as a strategic planning and project control toolset that project managers can use to dynamically and adaptively manage complex projects throughout their entire life-cycle. It is important to note, however, that integrating system dynamics modeling and simulation techniques into the entire project life-cycle would not work very well on novel projects. Projects where an organization has relatively decent experience would represent fertile ground for this type of new approach. For example, if an organization is experimenting with new technology, new business processes and/or a type of project it has never engaged in before, the integration of system dynamics models into the project management process will not only be lacking in value-added, it might actually detract from the end-result and hinder progress because there is no historical data available from which one could effectively initially calibrate the model. Projects that become somewhat repeatable (e.g. developing a management information system for a government organization) are good examples of where this integration would be applicable. Even though these projects are unique in nature, they should contain enough similar project variables and dynamics to calibrate a model with minor adjustments and account for some of the project's unique elements.

Lastly, more research and development could go into expanding the project dynamics covered in section III; taking some of those concepts one step further into the development of quantitative system dynamics stock and flow models, derived from the project behavior dynamics CLDs. These models could be used to produce adaptive strategic models and simulations, which would enhance both the project managers' skill and tool set, as well as provide a much need augmentation to traditional project management processes.

VI. Conclusions

The current state of information technology project performance is unacceptable and its failures are well documented in the literature. The discipline is poorly researched relative to other management disciplines and, coupled with the near exponential growth of technical and project complexity in the information technology industry, it is becoming increasingly evident that new systems-based models must be adopted for the field to advance and attempt to improve upon its mediocre performance rate. These new models do not necessarily need to replace traditional methods, but instead should augment them to produce a hybrid, or best-of-breed approach to managing complex IT/systems projects.

Because the nature and complexity of a project should dictate the method by which it is controlled, the project management discipline must continue to develop its operational and strategic methods and not continue to solely rely on reductionist methods that were developed more than half a century ago and apply more to the types of projects that originated out of that era. A new era of highly complex, deadline driven knowledge-work based information technology projects has developed in recent decades and dictates we take a holistic approach to project control, or suffer the consequences for ignoring to do so. This failure to properly adapt our methodologies and processes has been cited in the literature as both cause and effect for low project performance rates, the 90% syndrome or outright project failure. Adjustments to traditional models, the development of new models and the integration of both where appropriate, must be done to effectively account for the new complexities introduced by the global information age.

Moreover, project management professionals should seek to develop new competencies in systems-thinking approaches to project management, as well as the modeling and simulation tools needed to aid in that development. This expansion of core competencies would appropriately augment the current knowledge base that favors the types of reductionist approaches advanced by the Project Management Institute. The use of system dynamics benefits the discipline in many ways including, improvements in management training in the form of dynamic "flight simulator" models, management skill-set building, estimation and risk analysis techniques, project plans and network diagrams, what if" based scenario analysis used to craft effective intervention strategies, as well as a “best-of-breed” approach to management where the appropriate tools are used in the appropriate situations instead of being universally applied.

This area of research has just begun to tackle the project performance problem and will continue to search for new and improved methods for advancing the discipline. The value of system dynamics to project management is in its approach to effectively capture, communicate and strategically address project behavior issues using a systems approach that integrates both "soft" and "hard" variables into the model's equations. This is understandably a monumental task, especially when one considers the difficulty in adjudicating the validity of certain measures associated with soft variables; however, to ignore this task because debates over how to quantify certain aspects of a project can be contentious, is myopic in its approach and one might argue less effective because the model deliberately ignores the "reality on the ground," and consequently, is not representative of the system's true behavioral elements. System dynamics models applied to project management seek to be more inclusive of the project's true reality and, therefore, should be developed and matured into useful tools to improve performance that are widely disseminated and developed throughout the project management community.

VII. Bibliography

Abdel-Hamid, Tarek K. 1988. Understanding the "90% syndrome" in software project management: A simulation-based case study. *The Journal of Systems and Software* 8, (4): 319-30.

Abdel-Hamid, Tarek K., and Stuart E. Madnick. 1991. *Software project dynamics: An integrated approach*. New Jersey: Prentice-Hall, Inc.

———. 1990. The elusive silver lining: How we fail to learn from software development failures. *Sloan Management Review* 32, (1): 39-48.

———. 1989. Lessons learned from modeling the dynamics of software development. *Communications of the ACM* 32, (12): 1426.

Belassi, Walid, Alex Z. Kondra, and Oya I. Tukel. 2007. New product development projects: The effects of organizational culture. *Project Management Journal* 38, (4): 12-24.

Brooks, Frederick. 1975. *The mythical man-month: Essays on software Engineering* . enlarged edition 1995 ed. Reading, MA: Addison-Wesley.

Christensen, David S. 1993. An analysis of cost overruns on defense acquisition contracts. *Project Management Journal*(3): 43-48.

Cook, Rick. How to spot a failing project. in CIO.com [database online]. 2008 [cited August 5 2009]. Available from http://www.cio.com.au/article/203946/how_spot_failing_project?pp=1 (accessed August 5, 2009).

Dorsey, Paul. Top ten reasons why systems projects fail. in Harvard [database online]. Boston, 2000 [cited September 7 2009]. Available from http://www.hks.harvard.edu/m-rcbg/ethiopia/Publications/Top%2010%20Reasons%20Why%20Systems%20Projects%20Fail.pdf (accessed September 7, 2009).

Ford, David N., and John D. Sterman. 2003. The liar's club: Concealing work in concurrent development. *Concurrent Engineering: Research and Applications* 11, (3): 211-8, https://ceprofs.civil.tamu.edu/dford/dnf%20profesional/TheLiar%27sClubCERA-PUBLISHED.pdf (accessed October 1, 2009).

———. 2003. Overcoming the 90% syndrome: Iteration management in concurrent development projects. *Concurrent Engineering: Research and Applications* 11, (3): 177-86.

Galway, Lionel. 2004. *Quantitative risk analysis for project management: A critical review*. Santa Monica: RAND Corporation, , http://www.rand.org/pubs/working_papers/2004/RAND_WR112.pdf (accessed 5/29/2009).

Garvin, David. Learning in action. in Harvard Business School [database online]. Cambridge, 2000 [cited September 8 2009]. Available from http://hbswk.hbs.edu/item/1450.html (accessed September 9, 2009).

Garvin, David A., Amy C. Edmondson, and Francesca Gino. 2008. Is yours a learning organization? *Harvard Business Review* 86, (3) (03): 109-16.

Gido, Jack, and James P. Clements. 2009. . 4th ed. Mason, OH: South-Western Cengage Learning.

Gray, Clifford F., and Erik W. Larson. 2000. *Project management: The managerial process*. Boston: McGraw-Hill.

Hart, James D. 2008. *Discovering system dynamics in software engineering*. Matthews, NC: Software Process Dynamics, LLC.

Hass, Kathleen B. 2009. *Managing complex projects – A new model*. Vienna, VA: Management Concepts.

Holder, L. D., and Edward J. Fitzgerald. 1997. The center for army lessons learned: Winning in the information age. *Military Review* 77, (4) (Jul): 123.

Klein, Gary. 2007. Performing a project premortem. *Harvard Business Review* 85, (9) (09): 18-9.

Kotnour, Tim. 2000. Organizational learning practices in the project management environment. *The International Journal of Quality & Reliability Management* 17, (no. 4/5) (January 1): 393, http://proquest.umi.com/pqdlink?did=115716474&Fmt=7&clientId=50078&RQT=309&VName=PQD (accessed 8/06/2009).

Lewis, James P. 2008. *Mastering project management: Applying advanced concepts to systems thinking, control & evaluation, resource allocation*. New York, NY: McGraw-Hill.

Loch, Christopher, Svenja Sommer, Jing Dong, and Michael Pich. 2006. Step into the unknown. *Financial Times*, March 24, 2006, 2006, sec Mastering Uncertainty.

Lovallo, Dan, and Daniel Kahneman. 2003. Delusions of success. *Harvard Business Review* 81, (7) (07): 56-63.

Lyneis, James M., Kenneth G. Cooper, and Sharon A. Els. 2001. Strategic management of complex projects: A case study using system dynamics. *System Dynamics Review* 17, (3): 237-60.

Lyneis, James M., and David N. Ford. 2007. System dynamics applied to project management: A survey, assessment, and directions for future research. *System Dynamics Review* 23, (2-3): 157-89.

Matta, Nadim E., and Ronald N. Ashkenas. 2003. Why good projects fail anyway. *Harvard Business Review* 81, (9) (09): 109-14.

Maylor, Harvey. 2001. Beyond the gantt chart:: Project management moving on. *European Management Journal* 19, (1) (2): 92-100.

Meadows, Donella H. 2008. *Thinking in systems*. Thinking in systems., ed. Cannon Labrie. 1st ed. Vol. 1. Vermont: Chelsea Green Publishing Company.

Moyer, Don. 2007. What evidence? *Harvard Business Review* 85, (11) (11): 156-.

Project Management Institute PMI. 2008. *A guide to the project management body of knowledge (PMBOK guide)*. Fourth ed. Newton Square, PA: .

Rodrigues, Alexandre G. 2001. Managing and modeling project risk dynamics: A system dynamics-based framework. Paper presented at Fourth European Project Management Conference: PMI Europe, London.

Rodrigues, Alexandre G., and John Bowers. 1996. The role of system dynamics in project management. *International Journal of Project Management* 14, (4) (8): 213-20.

Rodrigues, Alexandre G., and T. M. Williams. 1998. System dynamics in project management: Assessing the impacts of client behaviour on project performance. *The Journal of the Operational Research Society* 49, (1) (Jan.): 2-15.

———. 1997. System dynamics in software project management: Towards the development of a formal integrated framework. *European Journal of Information Systems* 6, (1): 51-66.

Senge, Peter M. 1990. *The fifth discipline*. New York: Doubleday.

Shore, Barry. 2008. Systematic biases and culture in project failures. *Project Management Journal* 39, (4): 5-16.

Smith, Karl A. 2000. *Project management and teamwork*. Boston: McGraw-Hill.

Sterman, John D. 2002. All models are wrong: Reflections on becoming a systems scientist. *System Dynamics Review* 18, (4): 501-31.

———. 2000. *Business dynamics: Systems thinking and modeling for a complex world*. Boston: McGraw-Hill.

———. System dynamics modeling for project management. in System Dynamics Group [database online]. Boston, 1992 [cited November 8 2009]. Available from http://web.mit.edu/jsterman/www/SDG/project.html (accessed June 16, 2009).

System Dynamics Society. SystemDynamics.org. in System Dynamics Society [database online]. Albany, 2009 [cited 07/29 2009]. Available from http://www.systemdynamics.org/index.html (accessed 07/29/2009).

Taylor, Tim, and David N. Ford. 2006. Tipping point failure and robustness in single development projects. *System Dynamics Review* 22, (1): 51-71.

Upton, David M., and Bradley R. Staats. 2008. Radically simple IT. *Harvard Business Review* 86, (3) (03): 118-24.

USN, T. Laverghetta, and A. Brown USN. 1999. Dynamics of naval ship design: A systems approach. *Naval Engineers Journal* 111, (3): 307-23.

Williams, T. 2003. Learning from projects. *The Journal of the Operational Research Society* 54, (5) (May): 443-51, http://www.jstor.org/stable/4101731.

Williams, T. M. 1999. The need for new paradigms for complex projects. *International Journal of Project Management* 17, (5) (10): 269-73.

———. 1995. What are PERT estimates? *The Journal of the Operational Research Society* 46, (12) (Dec.): 1498-504.