Project Dynamics: Review of the Value of Systems Analysis Methodologies in Improving Information Technology Project Management

Okechi Geoffrey Egekwu[1] *[Timothy C. Delobe*[2]*]*

Abstract – Project failures in the Information Technology (IT) sector are 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. This paper highlights the role of system dynamics and other analysis tools 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 and others, will be synthesized (e.g. system dynamics methodology), and will suggest the need for more robust and value added project management approaches. Understanding the project dynamics will illustrate the complex interactions and feedback structures inherent in all projects, as well as seek to educate project managers on how to handle cause-effect relationships through the phases of a project. Furthermore, the research will illustrate problematic project dynamics, using various conceptual models, and suggest the need to integrate system analysis 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.
.

Keywords: Project Failures, Project Dynamics, Systems Analysis, Information Technology

THE PROBLEM

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 [1]. 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 [2, 3]. Some executives are even rejecting traditional organizational structures in favor of more complex models, like the matrix organization, or another type that Haas [1] 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.

[1] James Madison University, MSC 4102, Harrisonburg, VA 22807, egekwuog@jmu.edu

[2] James Madison University, MSC 4102, Harrisonburg, VA 22807, delobetc@dukes.jmu.edu

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 [1, 2, 3, 4, 5, 6, 7]. Some of the literature following this trend references *The CHAOS Report* by the Standish Group [8] 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. 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.

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-free 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 [5]. The report concluded that "the government got for its money less than 2 percent of the total value of the contracts". Project performance for software development has improved marginally since then but it is still deficient according to the Standish Group's Chaos report [8]. Abdel-Hamid and Madnick [5] 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 1 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 [5] 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.

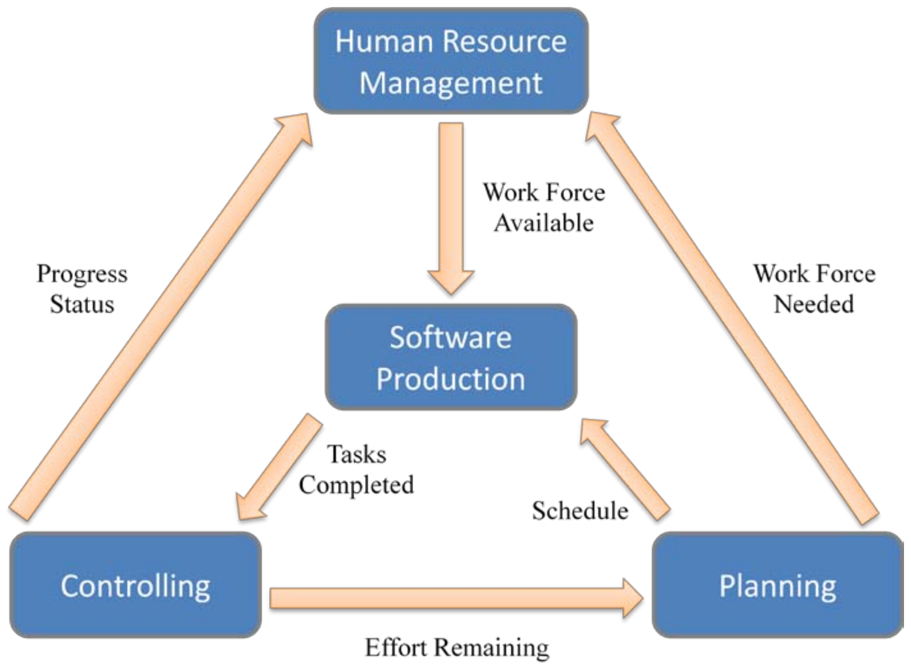

Figure 1: Fully integrated subsystem diagram for software development project dynamics (Abdel-Hamid and Madnick , [5]).

The *Human Resource* subsystem is comprised of functions such as hiring, training and assimilation. Abdel-Hamid and Madnick 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*.

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 that are witnessed in complicated project environments.

The *Software Production* subsystem comprises of four developmental 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.

As previously mentioned, this is one of the major differences between information systems/software development and traditional 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 [5] 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% .

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 [9, 10]. Sterman concisely defines the syndrome by stating that "a project is thought to be 90% complete for half the total time required" [10]. In more layman's terms, projects have an uncanny way of appearing to be on target and proceeding as planned until they approach 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.

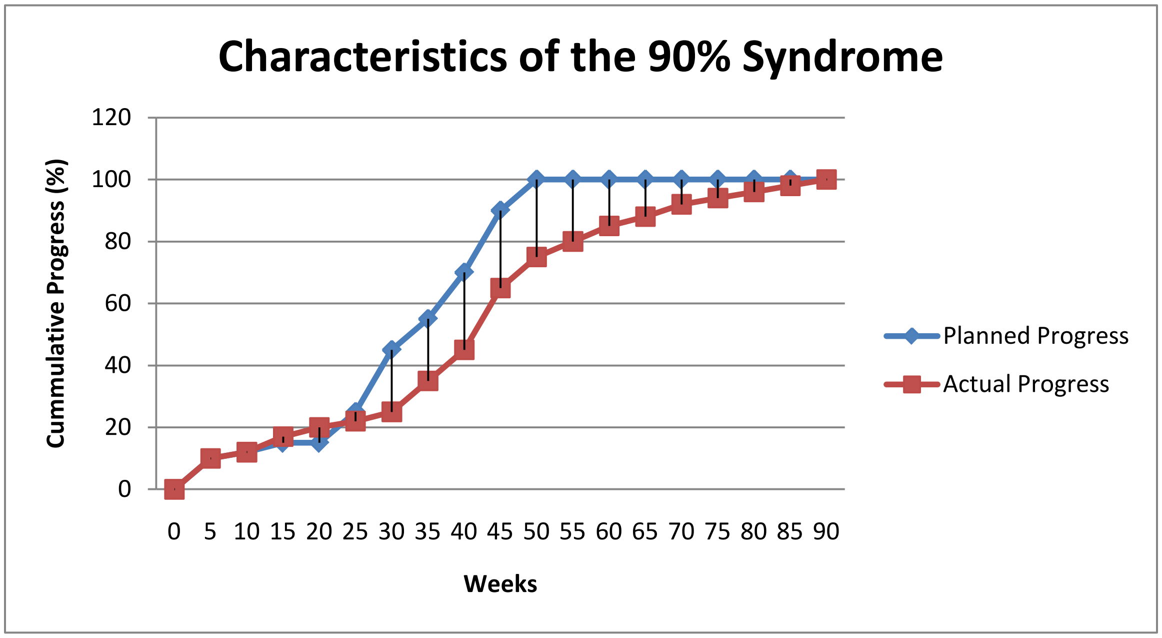

Figure 2: Classic S-shaped limits to growth representation of the 90% syndrome. Cumulative progress increases at a decreasing rate once the project nears 90% completion according to the original project plan (Ford and Sterman 2003).

There are several reasons cited in the literature for this project phenomenon depicted in figure 2. Abdel-Hamid and Madnick [5] 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 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.

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 [11]. 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 [12] categorize as "ripple" and "knock-on" effects. These actions often activate feedback dynamics, induced and exacerbated by management interventions, which can have the unintended consequences of producing an increase in the error rate, further reduced productivity and further delays.

The second factor Abdel-Hamid and Madnick [5] 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. 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 early in the project life cycle than problems with labor estimates and 2) because when size is underestimated, and subsequently detected in the early stages of the project, underestimation of man-days requirements are often revised appropriately. 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. 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, further degrading schedule/project performance.

In addition to Abdel-Hamid and Madnick's assertions on the causes for the 90% syndrome, Ford and Sterman [11] 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 tasks, as compared to working them in serial order.

Ford and Sterman [11] 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 [11]. 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.

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 practice widely used in system dynamics modeling, when devising policies to improve project performance in concurrent development projects. According to Ford and Sterman [13], "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, at a minimum, the following: 1) better estimation tools for both project size and man-days requirements; 2) a recognition that project estimates create project behaviors and vice versa; 3) 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; 4) 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 90% 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,". Lastly, the development and use of commercially available software project management simulation tools (e.g. system dynamics models) that are widely accessible, intuitive, extensible, and relevant must be incorporated in the decision-making process.

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.

AUGMENTING TRADITIONAL PROJECT MANAGEMENT METHODOLOGIES USING SYSTEM DYNAMICS LESSONS LEARNED

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 and the causes for the syndrome are extensive and complex in nature. While Adbel-Hamid and Madnick [5] assert that the main causes can be attributed to underestimation and the imprecise measurement of project progress, other studies by Ford and Sterman [11], Lyneis and Ford [12] and Hart [3] 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**.**

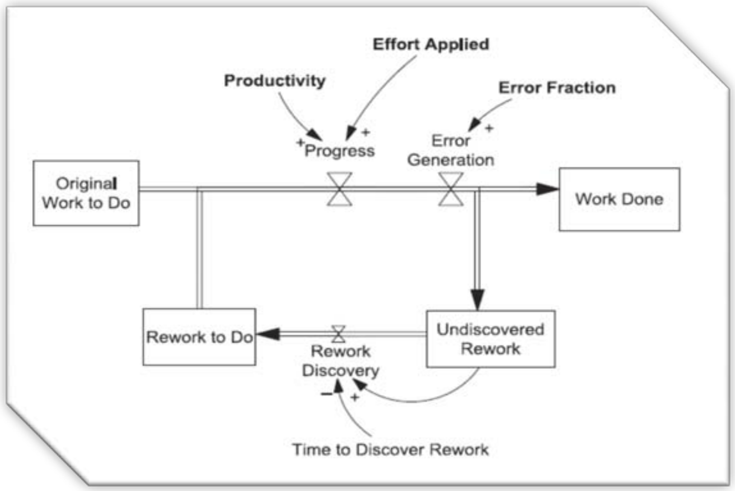

Figure 3: A typical development project rework cycle. Work packages move from the original work to do stock to the work done stock. In the process errors are discovered and sent back through the development process for "rework" (Lyneis and Ford 2007).

The various behaviors, interrelationships and causal connections that typify most software and web development projects are illustrated in figure 3. 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.

Williams [14] and Rodrigues [15] provide a rough outline for this new methodology – the integration of dynamic models with traditional project management methodologies, and other areas in the literature 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.

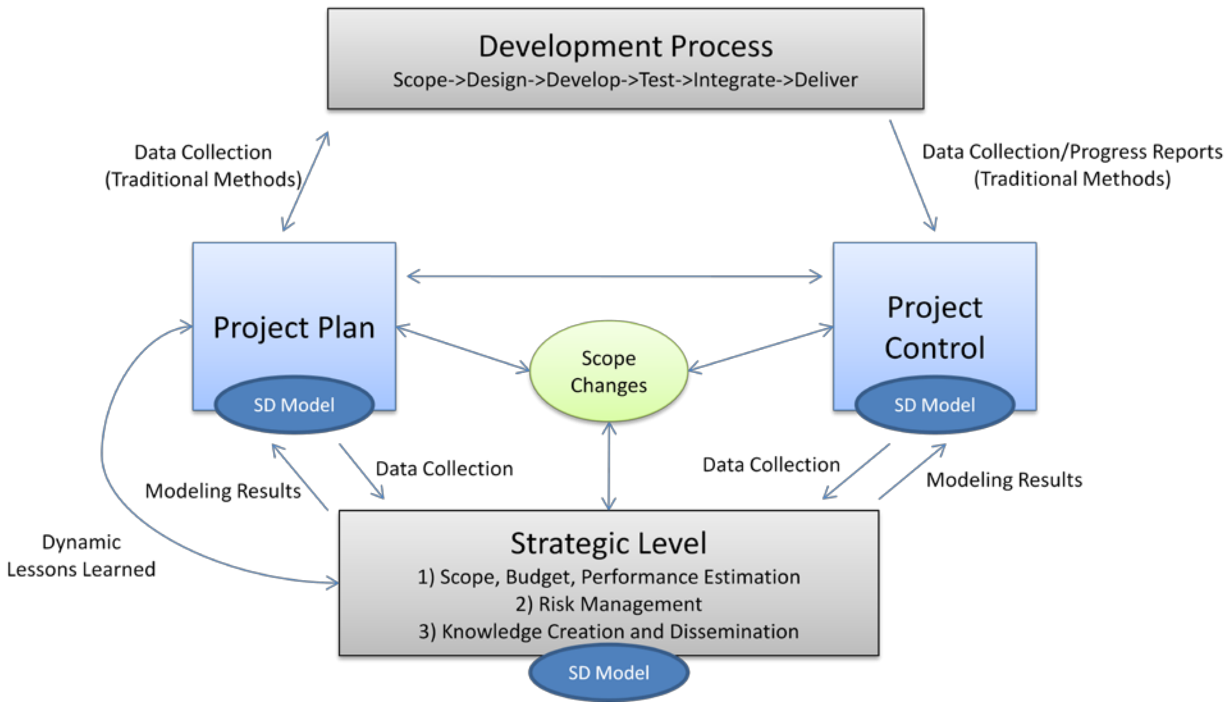
Development Process
Scope->Design->Develop->Test->Integrate->Deliver
Data Collection (Traditional Methods)
Data Collection/Progress Reports (Traditional Methods)
Project Plan
SD Model
Scope Changes
Project Control
SD Model
Modeling Results
Data Collection
Data Collection
Modeling Results
Dynamic Lessons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

Figure 4: 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. Adapted from [15].

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 4) 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.

Figure 5 illustrates some typical casual relationships in an IT project using system dynamics methodology. 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.

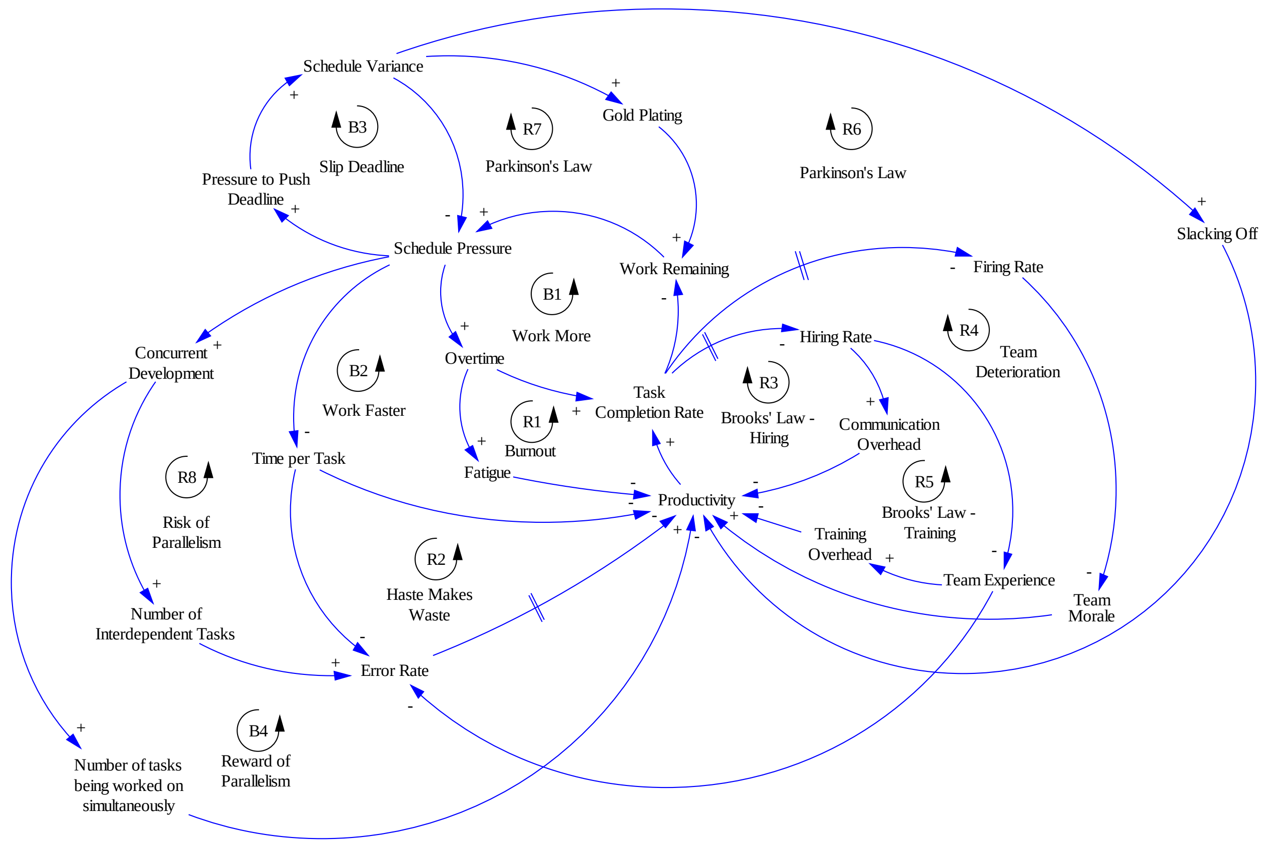

Figure 5: Conceptual model designed to illustrate typical systemic project behaviors and their causal connections Adapted from Sterman (2000), Lyneis and Ford (2007), Ford and Sterman (2003).

The 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. The conceptual model referenced in figure 5 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.

CONCLUSION

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 analysis methodologies to project management have the potential to effectively capture, communicate and strategically address project behavior issues using a systems approaches that integrate 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 analysis models (like systems dynamics technique) applied to project management seek to be more inclusive of a project's true reality and, therefore, should be developed and widely disseminated throughout the project management community.

References

[1] Hass, Kathleen, *Managing Complex Projects-A new Model*, Vienna, 2009.

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

[3] Hart, James D. *Discovering System Dynamics in Software Engineering*. Matthews, NC: Software Process Dynamics, LLC, 2008.

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

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

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

[7] 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).

[8] Standish Group. *CHAOS report.* Boston: The Standish Group International, Inc., 2009.

[9] Abdel-Hamid, Terek. Lessons learned from modeling the dynamics of software development. *Communications of the ACM* 32, (12): 1989, 1426.

[10] Sterman, John. 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).

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

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

[13] Ford, David N., and John D. Sterman. 2003. The liar's club: Concealing work in concurrent development. *Concurrent Engineering: Research and Applications* 11, (3): 2003, 217. https://ceprofs.civil.tamu.edu/dford/dnf%20profesional/TheLiar%27sClubCERA-PUBLISHED.pdf (accessed October 1, 2009).

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

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

Okechi Geoffrey Egekwu

Dr. Egekwu is a professor in the Integrated Science & Technology (ISAT) program at James Madison University, Harrisonburg, VA. He received his PhD degree in Industrial and Systems Engineering from University of Nebraska, Lincoln in 1992; and also has an MBA and B.S. in Chemical Engineering. He worked in industry (GM, Brunswick Defense, Alcoa Tech Center, etc.) for eleven years. Professor Egekwu is an active member of IIE, ASEE and SME.

Timothy C. Delobe

Mr. Delobe received his master's degree in Integrated Science & Technology from James Madison University in 2009. Professor Egekwu was Tim's thesis adviser, and this paper is adapted from Tim's thesis project. He manages *Interactive Systems Development* projects for the *Marine Corps News* in the Washington D.C. area.