2015 IEEE/ACM 37th IEEE International Conference on Software Engineering

Educating Software Engineering Managers – Revisited

What Software Project Managers Need to Know Today

Lawrence Peters
Universidad Politécnica de Madrid
Madrid, Spain
ljpeters42@ gmail.com

Ana M. Moreno
Universidad Politécnica de Madrid
Madrid, Spain
ammoreno@fi.upm.es

***Abstract*— In 2003, the original paper with this title was published as part of CSEET 2003. It focused on resolving communication issues between software project managers and developers and introduced a corporate strategy based means of evaluating software engineers. Now, more than a decade later, we could benefit from what we have learned in other fields about managing people involved in knowledge work and how to improve our success in software development. But are we? This paper is intended to present what Software Engineering students can be taught today that will help them to be successful as software project managers now and in the future. It is based on the premise that effective software project managers are not born but made through education.**

***Index Terms*—Software Project Management, Software Project Management Curriculum, Project Management.**

I. Introduction

In 2003, "Educating Software Engineering Managers" was presented as part of CSEET in Madrid, Spain. That paper focused on the unique personality characteristics of software engineers, introduced some communications issues and presented a new method of evaluating individual and team performance which was closely aligned with corporate strategy (SEM-Strategic Evaluation Method) [1].

Since then, tremendous changes have occurred in computing hardware technology, software development methods and other areas related to software engineering as well as management . More importantly, outside of software engineering, our collective knowledge regarding how people work, what motivates high technology workers and other matters of interest to software project managers has also expanded.

Today, in spite of all our achievements, we are still experiencing many of the problems cited in the first conference on Software Engineering [2]. Our pursuit of solutions in the technical realm since that time has not been as fruitful as we had hoped. In spite of evidence to the contrary [3], we continue to seek technical solutions rather than focusing on the role most responsible for project success – the software project manager [4]. A quick survey of the various software related conferences, post-doctoral research positions and articles in leading software publications reveals the near absence of attention to the subject of software project management.

As educators, this creates both an opportunity and a challenge. It is an opportunity in that we can play an important, positive role in shaping the future practice of software engineering. It is a challenge in determining what should be taught and how best to teach it. Some of the classic source materials [3,4] are a start but ignore many issues vital to success in software project management today. These success issues include strategy development and engagement, evaluating software engineers, creating and evaluating effective software development teams, controlling project costs and schedule, multiple risk assessment and mitigation methods, computing the confidence level in our cost estimates, and dozens of others. The remainder of this paper presents evidence that effective software project management education and training involves many topics outside of traditional software engineering subjects, most of which are anecdotal.

II. Technical Solutions from the Software Community

A lot has been written about Software Engineering's unfulfilled quest for "Silver Bullets" to solve our problems [5]. We keep contriving one programming method [6,7], development scheme [8] or lifecycle [9] after another in an attempt to address our productivity, cost, schedule and quality challenges. While all of the technical advances have had some positive effects, the actual productivity data have been disappointing. The improvement in productivity over the last 30+ years was less than one source line per person-month per year [10]. This improvement has been steady over the years but it has been linear. Even with the creation of newer software development methods, continued improvement in software development is unlikely to break out of this pattern and give us the quantum increase we need without revising our approach. The underlying assumption of many of the newer methods seems to have been that generating more code faster would improve success rates.

At the same time, we seem quite willing to go to any extreme to avoid facing up to the fact that without effective project management even the best staffed and sufficiently funded project might fail [11,12]. Similarly, we have consistently ignored the observation that poor project management can increase software project costs more than any other project cost factor. For example, up to 60% of software project cost can be attributed to personnel turnover [13]. The number one cause of personnel turnover is the project manager.

So, if we want to reduce software project costs, we should improve the way we manage people. Note, this does not relate to better technology in the strictest sense but better knowledge of how people work, can be motivated to perform at higher levels and how they might behave in crisis situations. That might not produce the quantum increase in productivity that we have been looking for but it is a start toward reducing costs significantly. So, if Software Engineering is to become a more viable profession with increased certainty of positive outcomes in software projects, more Software Engineers need to know how to plan, schedule and control software projects, but also acquire staff and motivate them to perform well.

Are we teaching our current and future software project managers properly to achieve those tasks successfully? Let's begin with a clear understanding of what software project management is.

III. Some Misconceptions about Software Project Management

Possibly due to a lack of knowledge regarding project management in general and software project management in particular, misconceptions exist.

A. We Are Not Alone

Part of our mindset in software engineering is that other fields have it easier. After all, they (e.g. construction) have been at it for thousands of years and the laws of physics they have to deal with have not changed. For example, the Romans built roads, bridges, aqueducts and other structures a couple thousand years ago many of which are still standing in places like Segovia, Spain, Rome, Italy and elsewhere. But even with all this experience, civil engineering projects still find estimating challenging (Table 1).

TABLE 1: Percentage Overruns by Project Category [14]

Project Type	**Average Overrun(%)**	**Overrun Range(%)**
Railways	45	7 to 83
Bridges and Tunnels	34	-28 to 96
Roads	20	-10 to 50

More recently, even with the aid of computers (or maybe because of them) there have been some spectacular failures and overruns in civil engineering and other construction projects. Assuming those are just anomalous events, construction still has its challenges. These can be particularly difficult when attempting to build a structure the likes of which has not been done before but this is not always the case. For example, at the time it was undertaken (1930), Hoover Dam was the largest, most challenging structure of its kind ever attempted. It had stringent scheduling, cost and performance restrictions incorporated into the contract, yet it was finished early and under budget. Years of planning, diligent management by a five company consortium and aggressive oversight certainly contributed to this unexpected positive outcome. They used the technology of the time consisting of slide rules, comptometers and adding machines so their success did not result from some technological breakthrough.

The lesson for software projects is that planning and scheduling are not a waste of time. Granted, the plan and schedule will undergo almost continuous change over the life of the project resulting from discoveries both positive and negative. Projects that have an acute lack of planning [15] experience costly miscues. One source of problems is the misconception that planning and scheduling are the same [16] – they are not. A schedule is a list of dates and corresponding events such as, "Project Kickoff – 01 February 2013, Project Status Meeting – 01 May 2013," and so forth. While a plan consists of a list of tasks and subtasks with start and end dates, personnel assignments, fixed and variable costs and so forth which, if successfully completed, ensure the project proceeds from one event to the next on schedule. While schedule and plan items are significantly different they do interact. Where do we teach Software Engineers how to plan their own efforts and when, in a leadership role, the efforts of others, correcting for biases that we now know exist [14], if not in Project Management classes?

B. Help from Other Disciplines

Software projects have been portrayed as unique, different from other creative endeavors, requiring software project managers to use, "a completely different set of tools" [16] than those used by project managers in other fields. Can this be true? We remain unaware of significant advances in individual and team management knowledge [17] that could help us improve in many areas. No matter what field they work in, all project managers face the same, often daunting task - create a product or service within a challenging budget and schedule, while meeting requirements that are often fluid using the human and other resources, funding and flow time that have been allocated. Just like the construction industry, unrealistic schedules, unexpected cost increases, changing requirements and regulations as well as other factors must be addressed as they occur. In other words, all project managers operate in a challenging environment they cannot control. The five basic functions of project management can be summarized in Planning, Scheduling, Controlling, Staffing and Motivating. These functions are not dependent on the industry, the type of system being developed or the technology employed, but are fundamental to project management and therefore to Software Project Management as well.

Similarly, a lot has been written about and promoted regarding software project cost estimation. From principles resulting in a Nobel Prize in Economics, optimistic biases present in all engineering estimates are recognized and documented. This optimism has been identified as an inherent human trait [18]. We now know how to remove these biases from software project estimates resulting in better estimates and fewer cost overruns [17]. It is so effective in comparison with more conventional approaches used in the past that it was officially endorsed by the American Planning Association with the recommendation that planners should "never rely solely on conventional forecasting techniques" [14,17]. Are our software engineering students being trained in the use of this more effective method for use in software project estimating?

C. *Help from another time*

For several years, the United States Department of Defense has required contractors to employ what many in the software engineering community see as an outdated project management status monitoring system that does not frequently apply to software projects – Earned Value Management. The fact that its origins date back to the late 19th century should not relegate it to the trash heap of management science. In fact, that long history, refinements and historical data provide us with some valuable information. For example, how many software project managers know that based on studies of hundreds of projects from high tech to low tech, if your project is 15% complete and over budget, its chances of getting back on track without altering the budget, the requirements, or the schedule are nil [19, 20]? Perhaps, if more Software Project Managers in this situation knew this single fact (and others) they would be even more diligent about not going over budget from day one. If they did go over budget, they could use this knowledge as an "early warning" that they should inform the project sponsor early on to provide relief in some area (e.g. requirements, schedule, budget). Lack of knowledge of this type (i.e. based on actual data and not opinion) can severely handicap Software Project Managers in their quest to bring projects in on time, on budget, meeting requirements, customer needs and satisfying perceptions of success.

IV. Software Projects Require Competent Project Managers

To some it appears that we do not need people to manage software projects. After all, we continue to create improved methods and industry hires and trains smart people to write code. But industry does not consistently train the person(s) who will be held responsible for the successful conduct of software projects – the software project manager [21].

The software project manager is the biggest single contributing factor in the success of a software project [21,22]. This is not because of the project manager's programming ability(s) but because of the need to have someone responsible for leading, organizing, planning and motivating a group of people into an effective team while communicating project status to Senior Management and the Client. Requiring a person to simultaneously serve as Software Project Manager and Software Engineer is not productive because managing and programming require very different, some would say diametrically opposed, mindsets [1,17]. Needless to say, this person's development activity will be interrupted by required participation in management meetings, personnel evaluations, status reports and presentations to the customer and future customers intermixed with having to comply with the syntax and semantics of the programming language, tracking down programming errors (bugs) and so forth. All of this results in a lack of focus and low performance in both areas.

However, this emphasis on technically oriented software project managers appears to be pervasive in our industry. In a worldwide survey of advertisements for project managers [23] out of 6 industry categories, the software industry was unique in its desire to hire project managers who had domain knowledge and skills (e.g. writing code) while the other 5 emphasized communication, leadership, team building and knowledge of management methods, cost control and management skills. How can we have good Software Project Managers if we do not recognize their required capabilities?

A. *Who Manages the Team?*

Whether working alone or in teams, software professionals need to know how to plan their work, setup a schedule, report status/progress and monitor the state of health of their efforts. They also need to learn how to work with others. Where will our Software Engineers and Software Project Managers learn how to assemble and successfully manage themselves and/or their teams if not in the classroom? Trial and error on the job training is what we are doing now. Team formation and evolution in a variety of industrial settings has been well researched and studied. There has been a considerable amount of research on teams doing knowledge work as well as specific studies regarding Software Engineering teams. Some of what these studies have shown would seem counter intuitive. For example, unless properly managed, when the team attempts to solve a complex problem, the sum of their experience and expertise will not be brought to bear on it. So, it is likely that without the proper training, Software Project Managers may not realize that some of their actions may contribute to cost and schedule overruns. Another misconception many software project managers have relates to team size. Most are well aware of Brooks' admonition to avoid adding people to a late project [3]. So, some software project managers believe that employing a larger software development team from the start improves a project's chances of being successful. The reality is it may not. This is referred to as, "The Team Sizing Fallacy" [24]. It has been demonstrated that larger teams become less effective as their size increases. How do we assemble an effective team in the first place? This is not as straightforward as it sounds [25]. NASA learned the hard way that assembling the best and the brightest into a team does not guarantee success [26] and may result in disaster. This phenomenon is called, "The Apollo Syndrome" [26]. Bringing the collective experience and skill of all the team members to bear on the complex problems associated with software development will most certainly reduce cost and improve the reliability of our estimates but achieving this requires an awareness of what it takes to effectively manage a software team to this end. If that end is "success," what success is varies from one group and individual to another [27]. What the project manager may see as success may be quite different from what the development team sees as success. Where will our Software Project Managers acquire this knowledge and the requisite skills to deal with it if not in the classroom?

B. Who Motivates the Team?

Motivated teams outperform unmotivated ones. The performance differences include both the quantity and the quality of the work delivered [28]. There are five factors associated with the tasks to which technology professionals are assigned that play a key role in determining their motivation level (Table 2). One additional factor was the abysmal performance of project managers in the high technology arena at providing feedback to the team, to individual members and the negative impact it has on productivity. It is uncomfortable for anyone to evaluate the performance of another. But that is the nature of being a software project manager and it is important to remember that feedback, even constructive negative feedback, is the number one most important value among all workers but ironically, it is, perhaps the least used because managers' number one value is money [29]. So managers give money to high performers when what the high performer wants most is a, "thank you" from their manager for their efforts without much fanfare. How many Software Project Managers are aware that the most common type of motivation (i.e. "Do this and get the reward") is relatively ineffective in "heuristic" work (e.g. software development) but most effective in algorithmic" (e.g. repetitive, assembly) work? Meaning, the incentive schemes commonly in use today have only limited value [30]. Worse, putting pressure on software teams to improve their performance may actually reduce productivity [31].

On the other hand, as we can see from Table 2, the differences between the priorities of the organization and the individual can create stress between them.

TABLE 2 – FRAMEWORK FOR WORK MOTIVATION [32]

Task Dimension	**Organization's Priority**	**Individual's Priority**
Skill Variety	Utilize the individual's skills	Learn new skills
Task Identity	Contribute to the organization	Contribute to advancement of the profession
Task Significance	Work on tasks important to the organization	Work on exciting projects from the standpoint of the profession
Autonomy	Understand and align with the organization's strategy	Be granted operational autonomy
Feedback	Subjective data and information processes	Objective data and information processes

In sum, are we teaching our prospective software project managers how to evaluate people to provide feedback, deal with this stress and achieve a motivated team?

C. Who promotes and guides communication among the team?

We continue to invest in training and research in programming and development technologies while industry demands for reliable cost estimates, delivery date certainty, and high quality continue to increase, remaining largely unfulfilled. Indicating we have not yet achieved effective software project management practice. Worse, this behavior is a symptom of a higher level problem. Some years ago, three psychologists published the results of studies which documented the inability of people to abandon a belief even in the face of overwhelming evidence that there was no data to support their belief(s) [33]. This appears to be a common human trait which the Software Engineering community is exhibiting individually and collectively. That is why we continue to pursue solutions to problems in Software Engineering in the technical realm even though we have 50 years of experience to the contrary. When will we change in our approach away from training in technology based solutions to project management methods?

This inability to seek other than technical solutions has resulted in a rapid progression from one technical solution to the next and so on. This behavior has been diagnosed by an expert in the field as symptomatic of attention deficit disorder [34,35]. He suggests that the fastest moving, most hard driving software companies have a relatively high proportion of people exhibiting symptoms of this disorder. They are not satisfied with the status quo and are always looking to make changes, improvements, and "breakthroughs." Even the next, more technologically advanced achievement is relatively quickly abandoned in favor of another somewhat unattainable one [36]. These characteristics are not exactly what a project manager would choose when trying to assemble a group of people into an effective team. There are ways to cope with this phenomenon [17] but first, the Software Project Manager needs to know it exists and be trained to cope with it.

Another symptom of this rapid abandonment of one method for the next is the issue of what constitutes success [27]. Project managers often see success as having a project finish on time, on budget and meeting or exceeding requirements - in other words, the business view of success. This is not always welcomed by software developers because they tend to want to be associated with technically superior software solutions and view the business side of the project with disdain. Thus, we have a dichotomy of opinion resulting in a natural communications gap. As pointed out many years ago [37, 38] communication is one of the manager's most important skills. Are we teaching communication skills (e.g. written expression, presentation and negotiation skills) to our software engineering students?

V. SOFTWARE PROJECT MANAGERS KNOW THYSELF

We now know why some project managers agree to an impossible schedule and/or budget [18]. Management researchers and psychologists have been trying to understand why people in all engineering professions do so poorly at estimating project costs and schedule. What they found was that there are some hidden forces at work. One is that decision making when one is at the lower levels of the organization is based primarily on facts and data and not on intuitive "feel" for the situation. As one moves higher in the power structure, reliance on facts and data declines and reliance on intuition increases to the point where reliance is solely on intuition [39]. Another finding, resulting in the 2002 Nobel Prize in Economics, explained how we make decisions with incomplete information, deal with uncertainty and agree to the impossible. This flawed decision making process is common to everyone. It can be overcome if we are aware of its existence and how to correct it [17]. What that research showed was without realizing it, we tend to be overly optimistic about our own abilities and those of our team while exaggerating the benefits to be derived from the project at hand. This causes us to minimize the risks while over estimating the benefits resulting in cost and schedule overruns. This is more than just an example of, "The Peter Principle" [40] wherein a person is promoted to their level of incompetence. It is a fundamental human trait. The more complex the project, the more pronounced the optimism. This appears to be part of software engineering's estimating problem? After so many years of trying to solve it with a broad range of methods [41], the answer would appear to be yes. This means the estimating methods we use, all of which ignore this factor, are also flawed. Shouldn't we be teaching our software engineers to correct for this?

VI. What Software Project Managers Need to Know

At this point it is fair to ask, "What should we be teaching Software Project Managers that will enable them to be more successful?" The short answer is, "Material very different from what is currently taught [42, 43] (if it is taught at all)." The longer answer involves exploring with students what the transition into management is about [44] plus eight key subject areas closely related to managing software projects, listed in Table 3. Some of these eight topics may seem far afield from the technical topics in many Software Project Management syllabi but they are based on a combination of input from what has been published regarding persistent, perhaps systemic problems in software project management [45], the authors direct experience and incorporation of synergistic methods published in various areas of project management, not just software engineering (see the references in Table 3). Much of this material is not presented in the Software Engineering Body of Knowledge [46] but has been incorporated from other disciplines.

The benefits of such an endeavor are not obvious. Let's look at an example of how such an approach could benefit the software project manager. For decades the software engineering profession has tried to "solve" the project estimating problem. In 2006, the American Planning Association recommended its members not use conventional estimating methods alone but always use Reference Class Forecasting [14] as well. That method resulted from a Nobel Prize winning result which explained why we do not estimate accurately and how we can correct the inaccuracy(s). Literally, none of the software estimation methods published in the last 50 years have addressed and/or solved this problem in this way. Shouldn't our software engineering students know how to more accurately estimate the costs and flow time for their work now that there is a better way to do it? Another example of subject matter that can benefit software engineering that did not originate within software engineering involves research and experimental results showing how adding people to a project does not have the benefits we expect [24].

TABLE 3 – Suggested Software Project Management topics

Topic	Sub-topic	Comments/Reference
Team Development	Forming successful software teams	Result of surveys [25]
	Interviewing Techniques	Perhaps most important [11,17]
	Psychological Compatibility	Individuals must work together [11, 17]
	Evaluating Teams and Individuals	The most difficult task for managers [11,17,32]
Communicat ion	Developing the Communications Plan	A coordinated effort [17,52]
	Written expression	Reports, emails are sometimes gibberish [52]
	Public Speaking and Presentation Skills	It can be a factor in success or failure [52]
Project lanning and Scheduling	Differences between Schedules and Plans	These are often confused to the detriment of the project [11,17]
	Pre and Post Project factors present in successful projects	Factors that make software projects more likely to succeed are known [45]
	Automated Planning and Scheduling Aids	Manually creating and maintaining these tools is not practical [17]
Complexity Management	The nature of project complexity	Understanding this helps to cope with and control it [17,47]
	Categories of Complexity	Not all issues equal – emphasis is on sociopolitical issues [17, 47]
	A Complexity Measurement Method	If we can measure it, we may be able to control /cope with it [17]
Strategic Planning	The Balanced Scorecard	Currently the most proven method [48]
	Strategy Maps	Effective means of implementing strategy and measuring progress toward achieving it [17, 49]
	SWOT	Tried and true means of developing the business case [17,50]
Estimating	Survey of Estimating Methods	Not all methods applicable to all projects [11]
	Overcoming Excessive optimism	Our view of our capabilities can be disasterous [14,17,18]
	Reference Class Forecasting	Method so effective, American Planning Association recommends its use in all cases [51,17]
Project Cost Accounting and Status Monitoring	Direct Costs	The obvious costs
	Indirect Costs	The hidden costs which kill projects
	Overhead	Hidden labor costs [11,17]
	General and Administrative	The costs of staying in business [11,17]
	Earned Value Management and Earned Schedule	Objective assessment of project Status in terms of money and flow time[17, 20]
Risk Management	Risk Assessment Methods	Survey of most effective ones [17]
	Applying Risk Mitigation	When and how often to assess risk using multiple methods [17]
	Risk Management Planning	A plan of action to prevent or deal with disaster [17]

Complexity management is a third example of the dichotomy between what is taught and what is needed in software engineering practice. Three types of complexity are recognized as being present in projects [47]: structural (involving size, scope and interdependence of tasks and personnel), sociopolitical (involving the importance of the project to the firm, people, politics within and outside of the firm, power, conflicting agendas within the stakeholder community), and emergent (uncertainty of the outcome, lack of experience with the technology or domain, lack of information, or some combination of these). In their study Maylor et al.[47] found that 68% of high technology project managers found sociopolitical complexity the most difficult to deal with while 87% of their training in complexity management focused on structural complexity.

A question that arises is how this knowledge can coexists with recent development approximations such as agile development. Agile development does not mean there is no need of management. Agile projects still need to be planned, scheduled, have a cost, multiple risks, very special teams that need to be managed and evaluated and might have an important complexity, similar to traditional projects. Therefore, practitioners that work on agile methods also need to know how to address these issues and also how to tailor them to fit into their agile processes.

Another relevant matter in this discussion is when the knowledge outlined above should be presented to students, that is, at the graduate or undergraduate level. The answer is, at both levels. Undergraduates because they will be asked how long they will take, how much it will cost, what will be done and when it will be finished. They will need the basics of scheduling and estimating. Students at the M.S. and PhD level will need the whole spectrum of skills from psychology to activity based costing. The differences between the two curricula are the details (M.S. = most detail, B.S. = less detail) and the scope, that is what topics will be included. M.S. graduates will be expected to develop and implement strategies consistent with overall corporate goals, be able to form effective development teams, evaluate team members, guide the work of the team, effectively manage complexity, and develop lesser performing team members into better performers.

VII. Closing Comments

Graduate and undergraduate software engineers need to be educated as to the importance of the software project manager as well as how to manage software projects by expanding the content of our course curricula. Certainly, there will be adjustments to the subject matter presented here to accommodate local, cultural [54] and generational [53] differences as well as new research in management science. Adaptations to that list should also be made to respond to the challenging needs of Software Project Managers today and in the future. The choice is ours – we can effectively achieve a quantum leap in software project performance or keep things as they are.

References

[1] Peters, L. J., “Educating Software Engineering Managers,” Conference on Software Engineering and Training (CSEET), Madrid, Spain, 2003

[2] Naur, P. and Randell, B. (Editors), “Software Engineering: Report on a Conference Sponsored by the NATO Science Committee,” Garmisch, Germany, 7 to 11 October, 1968, Brussels, Belgium, Scientific Affairs Division, NATO, January, 1969.

[3] F. P. Brooks Jr., “The Mythical Man-Month: Essays on Software Engineering,” Addison-Wesley, Reading, MA, 1995.

[4] Boehm, B., Software Engineering Economics, Prentice-Hall, Englewood Cliffs, N.J., 1981.

[5] Brooks, F. P., “No Silver Bullet – Essence and Accident in Software Engineering,” Proceedings of the IFIP Tenth World Conference, Elsevier Science, 1986.

[6] J. Shore, “The Art of Agile Development,” O’Reilly Media, October, 2007.

[7] L. Lindstrom and R. Jeffries, “Extreme Programming and Agile Software Development Methodologies,” CRC Press, 2003.

[8] McConnell, S., “Managing Technical Debt,” White Paper, Construx Software Corporation, June, 2008.

[9] Boehm, B., “A Spiral Model of Software Development and Enhancement,” Computer, May, 1988, pp. 61-72.

[10] R. Jensen, “Don’t forget about good management,” CrossTalk Magazine, p. 30, August 2000.

[11] L.J. Peters, “Getting Results from Software Development Teams,” Microsoft Press - Best Practices Series, Redmond, WA, 2008.

[12] P. Ghazi, A.M. Moreno, and L. Peters, “Looking for the Holy Grail of Software Development,” IEEE Software, January/February, 2014 pp. 13-16.

[13] E. Cone, “Managing that churning sensation,” Information Week, May 1998, No. 680, pp. 50-67.

[14] B. Flyvberg,“From Nobel Prize to Project Management: Getting Risks Right,” Project Management Journal, Volume 37, Number 3, pp. 5-15, August, 2006.

[15] H. Thanhaim, “Team Leadership Effectiveness in Technology Based Project Environments,” IEEE Engineering Management Review, Vol. 36, No. 1, 2008, pp. 165-180.

[16] S. McConnell, “The software manager’s toolkit,” IEEE Software, July/August 2000.

[17] L.J. Peters, Managing Software Projects: On the Edge of Chaos - From Antipatterns to Success, Kindle eBook, 2015.

[18] D. Lovallo and D. Kahneman, “Delusions of success – how optimism undermines executives’ decisions,” Harvard Business Review, July 2003

[19] C.P. Beach, “A-12 Administrative Inquiry,” Office of the Under Secretary of Defense for Acquisitions, Washington, D.C., November 28, 1990.

[20] Q.W. Fleming and J.M. Koppelman, “Earned Value Project Management – Fourth Edition,” pp. 39-40, Project Management Institute, Newtown Square, Pennsylvania, 2010.

[21] J.V. Farr and D. M. Brazil, “Leadership Skills Development for Engineers,” IEEE Engineering Management Review, Volume 21, Number 1, pp. 13-22, March, 2009.

[22] Weinberg, G., Quality Software Management, Volume 3: Congruent Action, Dorset House Publishing, New York, NY, June 1994.

[23] M. Chipulu, J.G. Neoh, U. Ojiako and T. Williams, “A Multidimensional Analysis of Project Manager Competences,” IEEE Transactions on Engineering Management, (in press – IEEE Transactions on Engineering Management, Volume 60, Number 7, pp. 506-517.

[24] B. R. Staats, K. L. Milkman, C. R. Fox, “The team scaling fallacy: Underestimating the declining efficiency of larger

teams," Organizational Behavior and Human Decision Processes, 118 (2012), pp. 132-142.

[25] S. J. Chen and L. Lin, "Modeling Team Member Characteristics for the Formation of a Multifunctional Team in Concurrent Engineering," IEEE Transactions on Engineering Management, 15(2), pp. 111-124, 2004.

[26] R. M. Belbin, "Management Teams - Why They Succeed or Fail," Butterworth Heinemann, London, 1996.

[27] K.R. Linberg, "Software Developer Perceptions About Software Project Failure: A Case Study," The Journal of Systems and Software Volume 49, pp. 177-192, Elsevier Publishing, 1999.

[28] B. Staats and D. Upton, "Lean Principles, Learning, and Software Production: Evidence from Indian Software Services," Working Paper 08-001, Harvard Business School, 2007-2009.

[29] Families and Work Institute, "National Study of the Changing Workforce," Published by The Families and Work Institute, New York, NY, 1993.

[30] R. M. Ryan and E. L. Deci, "Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions," Contemporary Educational Psychology, Volume 25, pp. 54-67, 2000

[31] H. K. Gardner, "Performance Pressure as a Double Edged Sword: Enhancing Team Motivation While Undermining the Use of Team Knowledge," Working Paper, 09-126, Harvard Business School, January, 2012.

[32] Katz, R., "Motivating Technical Professionals Today, " IEEE Engineering Management Review, Volume 41, Number 1, March, 2013, pp. 28-38.

[33] Festinger, L., Reichen, H. and Schachter, S., "When Prophecy Fails," Martino Books, Eastford, CT, 2011.

[34] J. J. Ratey, live interview on KUOW radio, Seattle, Washington, 26 April, 2001.

[35] E. M. Hallowell and J. J. Ratey, "Driven to Distraction: Recognizing and Coping with Attention Deficit Disorder from Childhood through Adulthood," Simon and Schuster, New York, New York, 1995.

[36] M. Lewis, "The New New Thing," Penguin Books, New York, New York, 2000.

[37] R.C. Townsend, "Up the Organization: How to Stop the Corporation from stifling people and strangling Profits," New York, NY, Alfred Knopf, 1970.

[38] Dow, W. and Taylor, B., "Project Management Communications Bible," Wiley Publishing, Indianapolis, IN, 2008

[39] Merrington, C., "Why do Smart People Make Such Dumb Mistakes?," Academy Press, N.Y., NY, 2011

[40] Peter, L. J. and Hull, R., "The Peter Principle: Why Things Always Go Wrong," Harper, N.Y., NY, 2009.

[41] D. F. Rico, "Short History of Software Development Methods," posted on the web, 2010.

[42] T. C. Lethbridge, R. J. LeBlanc, Jr., A. E. Kelley Sobel, T. B. Hillburn, and J. L. Diaz-Herrera, "SE2004: Recommendations for Undergraduate Software Engineering Curricula," IEEE Software, November/December 2006, pp. 19-25.

[43] Stevens Institute of Technology, "Curriculum Guidelines for Graduate Degree Programs in Software Engineering," as part of the Integrated Software & Systems Engineering Curriculum Project, Version 1.0, September 30, 2009.

[44] T. Tarim, "Making a Transition from Technical Professional to Project Manager," IEEE Engineering Management Review, Volume 40, Issue 3, 2012

[45] Silva, P., Moreno, A., and Peters, L., "Software Project Management: Learning from Our Mistakes," IEEE Software, Voice of Evidence column, March/April, 2015.

[46] IEEE Computer Society, "Guide to the Software Engineering Body of Knowledge," IEEE Computer Society, January,2014.

[47] Maylor, H.R., Turner, N.W., Murray-Webster, R., "How Hard Can It Be? – Actively Managing Complexity in Technology Projects," Research-Technology Management, July-August 2013, pp. 45-51

[48] Kaplan, R. S. and Norton, D. P., "Linking the Balanced Scorecard to Strategy," California Management Review, Vol. 39, No.1, Fall 1996.

[49] Kaplan, R. S. and Norton, D. P., "Strategy Maps: Converting Intangible Assets into Tangible Outcomes," Harvard Business Review Press, Boston, Massachusetts, 2004

[50] Humphrey, A., "SWOT Analysis," Stanford Research, 1970

[51] Flyvberg, B., Holm, M.K.S., Buhl, S.L., "How (In)Accurate are Demand Forecasts in Public Works Projects? The Case of Transportation," Journal of the American Planning Association, vol. 71, no. 2, Spring, pp. 131-146.

[52] Project Management Institute, "The High Cost of Low Performance: The Essential role of Communications," Project Management Institute, May, 2013.

[53] Crowder, J.A. and Friess, S., "Agile Project Management: Managing for Success," Springer, 2014

[54] Kankanhali, A., Tan, B.C.Y., Wei, K. and Holmes, M., "Cross-Cultural differences and information systems developer values," Decision Support Systems, Volume 38, No. 2, pp. 183-195, 2004.