Toward A Better Software Project Manager

Lawrence Peters, Software Consultants International Limited, Auburn, Washington, United States
Lecturer, Universidad Politecnica de Madrid, Madrid, Spain

Abstract. The software engineering profession has experienced several decades of almost unnoticed successes and famous failures. Over that time, its accomplishments have far exceeded what the founders of this profession could have foreseen. One aspect of software engineering that has gone almost unnoticed is the necessity for competent software project managers. Recent studies have shown that the software project manager plays a much more significant role in the success of software projects than previously thought. This article examines the role software project managers play in the development of software systems, explains why this has been largely overlooked, and proposes that the practice of software project management has always been and continues to be critical to the success of software projects today and into the future

Introduction

All significant engineering efforts require someone to manage them. Fven from their simple beginnings, software development efforts have needed someone responsible for ensuring the work was completed satisfactorily. The common term for the person responsible for such efforts is "project manager."

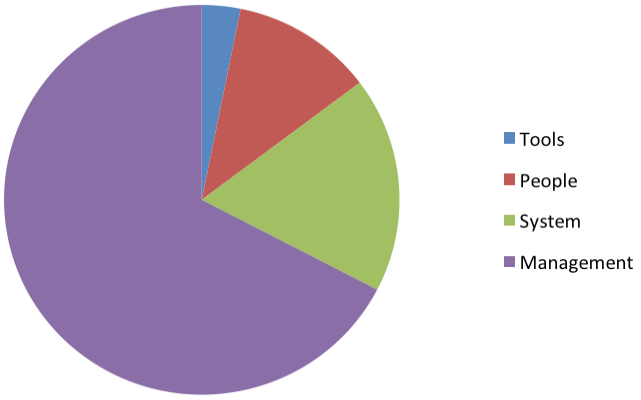


Figure 1: Relative effects of factors involved in software projects

Early in the evolution of the software engineering profession, projects were, as compared with today's efforts, relatively simple, standalone efforts to automate some aspect of a business or government activity. But as time went on, these single-purpose systems became integrated and evolved into complex interactive systems operating in real time, simultaneously serving the needs of thousands of users online. That complexity did not occur overnight, but it occurred rapidly enough to overwhelm software project managers who were largely untrained in project management. Many had created their own approaches to managing software projects, but even simple projects often failed to meet budget and schedule constraints.

This caught the attention of NATO, resulting in the first international conference on software engineering [1]. While software engineers were trained in a broad range of methods and techniques [2] [3], software project managers were left alone to cope with this increased demand for speed, ease of use, security, and other system characteristics with the added pressure to bring the project in on time, on budget, and sufficiently functional and reliable. Additionally, some people did not view the role of the software project manager as fundamental to successful software projects. Recognizing that the role of the software project manager is critical to the successful execution of a software project may be fundamental to the advancement of the profession of software engineering.

Competent Software Project Managers

From the beginning of the software engineering profession, it was assumed that "a cadre of competent project managers" existed to manage software projects, but they did not. [4] In developing his book on software engineering economics, Boehm had omitted software project management factors from the "constructive cost model" (COCOMO) because he assumed that "project management was uniform, constant, and good" [5] when, in fact, it was not. The observation that software project management has been uniform, not good and practically unchanged since 1970 appears to be true today. [6]

The Impact of the Software Project Manager

Two decades ago, Weinberg showed that the impact of the software project manager on a software project ex-

ceeded that of all other groups combined. He did this by combining the various cost driver groups in COCOMO [7]. See Figure 1 (based on Jensen's "Don't Forget About Good Management"). [6]

That work seems to have gone unnoticed for many years. More recently, a study by IBM [8] found that 54 percent of project failures were attributable to poor project management, while three percent were due to technical challenges. In spite of these and other qualitative and quantitative results highlighting the importance of the software project manager, today's software engineering curricula, conferences, publications and teaching position announcements rarely mention software project management as a subject of interest. When undergraduate and graduate software engineering curricula do include software project management as a course, it is frequently listed as optional.

In spite of the many cost overruns, delayed deliveries and mediocre results of some software systems, others have been successfully delivered by teams of software engineers led by largely untrained software project managers. [9] The problem today is that the increased complexity and importance of these software systems means that conducting software development in a "business as usual" manner — i.e., with uncertain delivery dates, cost overruns, poor quality, and poor maintainability — is likely to lead to a self-limiting future in which new systems and system upgrades are avoided or curtailed altogether. If the software engineering profession is to have a successful future, it needs more well-trained, competent software project managers. Perhaps more importantly, software engineers need to be made aware of the importance of the software project manager and the value software project managers bring to software projects.

Improving the Status Quo

In software engineering, we are trying very hard to change the status quo. However, the areas we are focusing on are not those that can provide the highest return on investment. For example, programming methods like Agile and Extreme have improved many aspects of software engineering. But as Weinberg observed [7] and, more recently, the work by Gulla demonstrated, [8] these are at best secondary effects. For example, from 1960 to 1990, the focus was on programmer productivity. It increased at a linear rate of just one source line per programmer month per year. [6] We have no reason to believe that the linear increase over that period has changed significantly since then. While that modest increase is positive, it occurred during a period when dozens of computer-aided software engineering tools, as well as analysis, design and programming methods, were published and adopted by the software engineering community. [2] It should also be noted that producing large amounts of source code has not always been the problem. Obviously, the biggest improvement in software engineering project performance, and the highest leverage area, is software project management. But how can this be accomplished?

A Better Software Project Manager and Software Project Management

Effective software project managers are not born; they are made through education, experience, mentoring and other related means. Our profession's lack of recognition of the importance of software project management is reflected in many ways. For example, the Software Engineering Body of Knowledge (SWEBOK) [10] treats software project management in the same way it treats other, often esoteric, topics in software engineering. If we really recognized the impact the software project manager can have on a project, SWEBOK would devote much more space to it or, perhaps due to its importance and the significant differences between software project management and software engineering, software project management should have its own standalone body of knowledge (e.g., SWPMBOK). [17] Recently, a software extension to the Guide to the Project Management Body of Knowledge, (PMBOK®) [12] was published in an attempt to remedy this situation. [13] But it is an extension, not a standalone document that focuses squarely on software project management. This extension requires the user to continually reference PMBOK to obtain the needed information — assuming that the information is present, which it may not be because information is generalized and covers a broad range of industries. A more specific discussion of what software project managers need to know in order to be successful has been developed and published. [14] Based on this new reference, the extension does not completely address the knowledge, methods, techniques, and data software project managers need to understand in order to be successful.

Software project management has little to do with programming and a great deal to do with what are commonly referred to as "soft" skills (i.e., communication, staffing, motivating, coping with complexity, risk management, and personnel issues) that software project managers need in order to be successful. The real question today is what resources — e.g., referential documents, university training, and professional development courses —exist to assist current and future software project managers in acquiring the knowledge and skills they need to be more consistently successful? Granted, not all software engineering professionals may want to become software project managers, but many will. When they do, they will need to know what software project management is about.

The most important fact they need to accept is that software project management does not involve the same domain of technology as programming. It requires a different mindset. The software project manager is not developing the software, but is instead working with the development team to create a plan by which the team will develop the software. The project manager re-plans as needed, monitors progress, and reports the project's status to key stakeholders while working to remove obstacles that prevent the software engineering team from working at their highest performance level. It mostly involves the personnel and business end of the software engineering profession. Failure to accept this fact has probably contributed significantly to the number of software projects that have failed.

As one noted software professional observed, "Management, not technology, determines success." [15] Also, the education side of this problem will help software engineers to better understand the importance of software project management and the value that software project managers bring to software projects. But software project managers do not need to be among the best software engineers in order to be effective. [16] This is because the best software engineers often become frustrated software project managers who have little patience with those who are less proficient than they are and may not mentor others to reach their full potential. This often results in a frustrated software project manager who reverts to being a software engineer, leaving the project without a manager. The dangers involved in putting high performers into management were documented long ago, but the message has still not had widespread acceptance. [17] [18]

Who Should be Educated?

If we look at companies engaged in software engineering projects from a system-level point of view, the delivery side is composed of software engineers, software project managers and senior managers. This last group, senior managers, are the ones who decide who becomes a software project manager and reviews their performance. A few decades ago, a very successful chief executive officer of major international corporations wrote an advisory against putting high performers into management. [16] [18] He pointed out that putting the best at performing certain tasks into management was not advisable. This is because doing so reduces the overall performance of the group they were in, and this new manager is likely to be frustrated by the non-technical problems that he or she would be required to solve every day — the most challenging of these being personnel-related problems. [9] Compounding this phenomenon are the various perquisites attached to being in management that can attract people into management for the wrong reasons (e.g., better pay, a better office, status within the company, a preferred parking space). But this sage advice has been largely ignored by the corporate world. The result is, in part, what we have today. Therefore, three groups must be educated:

—Software engineers. In most aspects, we are doing a good job of training software engineers. They are being taught software design, quality-oriented programming practices, testing methods and strategies, how to address security issues, and nearly every other aspect of software engineering. They are being exposed to new concepts and nuances, many of which are being published at a phenomenal rate. However, we are failing to accurately communicate the role of the software project manager to software engineers. The role of software project manager is not technical in nature but is oriented toward organization, planning, scheduling, controlling, staffing and motivating. [19] [20] [21] Many software engineers still believe that the software project manager should be the most technically astute member of the team. Their training does not accurately portray what software project managers do or how this role differs significantly from developing code. The sports analogy of manager versus player is an appropriate one, [22] but it is not often conveyed in software engineering courses. Training in this area will at least improve the software engineers' understanding of what they are getting into should they pursue a position in software project management.

- Software project managers. Most software project managers are not trained in appropriate management skills. [9] [21] This leaves them with little or no basis for actions and decision-making as managers of software engineering teams. This has resulted in the creation of a large number of "anti-patterns." An antipattern is a management action taken to solve a problem that actually makes things worse. [23] These are difficult to remove from the manager's lexicon of problem solutions, since many were created by the manager or suggested by a colleague who also created antipatterns. These actions

Senior Manager Rank	Project Manager Rank	Problem / Issue
1	10	Insufficient front end planning
2	3	Unrealistic project plan
3	8	Project scope underestimated
4	1	Customer/management changes
5	14	Insufficient contingency planning
6	13	Inability to track progress
7	5	Inability to detect problems early
8	9	Insufficient number of checkpoints
9	4	Staffing problems
10	2	Technical complexity
11	6	Priority shifts
12	11	Personnel not committed to plan
13	12	Uncooperative support groups
14	7	Sinking team spirit
15	15	Unqualified project personnel.

Table 1: Differing perceptions of causes of project failure [27]

can effectively sabotage a software project. [21] One antipattern that demonstrates a clear lack of understanding of personnel issues is a common belief relating to software quality. More than one published paper has tacitly assumed that producing quality results is more costly than producing poor or mediocre results. [24] What this concept ignores is the fact that everyone wants to be associated with quality results and is motivated to achieve them, [25] which results in higher productivity. As one author put it, "Quality is free." [24] [26] There are many other misconceptions related to software development, but this one clearly demonstrates both a misunderstanding of what drives software engineers to excel and the assumption of a negative impact without data to support it. Since software will never be perfect, the software project manager working with the rest of the team plays a critical role in determining whether the system is good enough to ship. In addition to antipatterns, there are many sources of communication issues between the software project manager and senior management — the people who put this person into the position of software project manager. These issues have been documented and are presented in Table 1. [27]

Factor	Manager's Importance Rank	Non-Manager's Importance Rank
Salary	1	5
Job Security	2	4
Promotion/Growth Opportunities	3	7
Working Conditions	4	9
Interesting/Challenging Work	5	6
Personal Loyalty to Workers	6	8
Tactful Discipline	7	10
Appreciation for Work Done	8	1
Help with Personal Problems	9	3
Being in on things	10	2

Table 2: Value system differences between managers and non-managers [33]

- Human resources professionals. In most firms, the path to becoming a software project manager is neither clear nor well documented. [9] This can result in software engineers who wish to become software project managers being frustrated by the happenstance nature of such a transition, while those who may not have sought to become software project managers have this transition thrust upon them. Others may be attracted by the perquisites but not really motivated to pursue this change in their career path. [14] [21] These circumstances have resulted in what we have today — software project managers of varying quality, inconsistency in the success of software projects, general lack of knowledge of the role of a software project manager, and a lack of recognition of their value to a project. [8] [9]

The Consequences of Our Inattention

In these situations, the software project manager often reverts to being a software engineer, effectively leaving the project without a manager. This leads to predictable project failure. A brief examination of the few software project manager courses reveals their content is mostly focused on software development, not on management issues such as personnel management, [9] negotiation, risk management, effective cost and schedule estimating methods, communication, planning methods, and so forth. As an example of personnel issues, consider this: software project managers and non-managers have very different value systems that lead to significant communication issues. Table 1 [27] summarizes these issues. At the very top of the list is what may be the most serious misconception on the part of software project managers. In fact, people do not work for money. Their motivation goes much deeper than that. Research has determined that people work for self-fulfillment, self-realization, and other factors. [28] [29] [30] Money is not a motivator in knowledge work. Money is a motivator in repetitive, assembly line work like one would find in a factory. The point is that value systems differences between software project managers and software engineers create a climate within which communication is inhibited. This predictably lowers productivity. Another significant example is "appreciation for work done." Software project managers are often uzzled by the resignation of a top performing software engineer who receives an extraordinary salary increase. If the recipient did not view it as a "thank you" for their efforts, the money would mean very little. Thus, we have a "reward paradox" wherein the most expensive reward is the least effective and the least expensive reward — a simple "thank you" — is the most effective. [31]

- Almost any improvement in what we teach software project managers will have a positive impact on projects.
- Senior managers. This group has been mostly ignored by our educational efforts. Due to their position in their respective companies, they tend to rely more on their own beliefs and conclusions than facts and data. [32] Stated another way, when at the lower levels of the power structure in an organization, one bases their decisions almost solely on facts and data. As one advances higher in the power structure, reliance on facts and data diminishes and decisions are based almost solely on intuition. The group at the top, senior managers, is responsible for making the most accomplished software engineers into software project managers, often with disastrous results. [9]

Compounding this situation is a natural communication gap between senior managers and project managers. (Table 2) [33] This may disappear as more software project managers get

into senior management positions, but for now, it needs to be contended with. What this group needs to accept is an overhaul of the way in which people become software managers and what perquisites (if any) are appropriate for such positions.

A Natural Communications Gap

A software project manager's actions are based on his or her value systems. These value systems have been documented by different authors over several decades. [33] (See Table 2.) Note how different the values are between the two groups. The values of each group have changed little over the years and vary slightly from one researcher to the next. One area that highlights misconceptions on the part of many software project managers is salary. As a reward for work well done, software project managers will award an outstanding software engineer with either a salary increase or a bonus because salary is the manager's most important value. Unfortunately, the non-manager's most important value is appreciation for work done. This could be achieved via a simple "thank you" conveyed in private by the software project manager. [31]

Another challenge for software project managers is managing the differences in perceptions of what caused a project to fail. In Table 1 we see several obvious sources of problems for the software project manager. Remember, it is the senior management team that will determine the software project manager's future with the company.

One example of the communication issues in this domain is the perceived reason for project failure. The software project manager lists the most serious issue causing project failure as changes in the customer and/or the customer's management team. Changes of this sort are inevitable but are still listed as causes of failure. Unfortunately, this is consistent with a prevalent attitude regarding project failure, which is that much of what causes a project to fail is ambiguous. This usually allows the manager of a failed project to shirk responsibility for the failure. [34] This has the effect of preventing the software project manager from learning from failure. This phenomenon may explain why software engineering seems to continue to experience serious project cost and schedule overruns for so many years — we do not learn from our failures.

Another important aspect of the contents of Table 2 relates to the nature of these factors. Regardless of how they are ranked, none are, strictly speaking, technical. They break down into the following categories:

- Planning and scheduling. Contrary to the opinion of some, [35] planning and scheduling are not the same activity. They are closely related, but are not the same. Planning focuses on developing a list of tasks and subtasks to be performed and who will perform them. Scheduling deals with the order in which the tasks and subtasks must be performed, the length of time each will take and the start and end date of each. To most software project managers and software engineers, planning and scheduling are uncomfortable tasks. A quote by Sir John Harvey Jones captures this discomfort: "People don't like to plan — planning is unnatural — it is far more fun to just do. And the nice thing about just doing is that failure comes as a complete surprise, whereas if you have planned, the failure is preceded by a long period of despair and worry."
- Controlling and tracking progress. A concern most software project managers have is runaway development. Unlike other engineering professions (e.g., civil engineering), software engineers can begin developing with a plan, schedule, design or even a set of requirements. This has caused many software project managers to develop and enforce a software development process that includes project reviews, milestones and GO/NO-GO decision points. "Earned value management" [36] and "earned schedule" [37] are often used in this regard.
- Personnel management and motivating. Most software project managers agree that personnel-related tasks are the most difficult to deal with. [9] There are many reasons for this, including the fact that work holds such an important place in the human psyche [28] [29] [30]. In addition, high technology workers like software engineers

are motivated to work on projects that advance the state of the art and provide them with talking points with colleagues that indicate they are exceptionally skilled. [9] This creates a particularly challenging situation for software project managers, since it is unlikely that all software projects they will manage — as well as the roles within those projects — will advance the state of the art. Rotating teams among projects can provide a means of ensuring that the software engineers do not continually work on projects they consider uninteresting and unchallenging.

• Personnel and team composition. Selecting the "right" team members is not nearly as easy and straightforward as it sounds. Creating a team of the best software engineers does not often lead to successful project completion due to a phenomenon known as "The Apollo Syndrome." [38] It refers to documented evidence that the most highly skilled technical people rarely work together as a mutually supportive team. Even interviewing can be difficult due to company policies, various labor laws and, sometimes, the belief systems of potential candidates. This often leads software project managers to select people who think the same way the manager does. This results in a team that shares the same mistaken beliefs about software project execution as their project manager, which results in project failure. [19]

• Communications with team, client, senior managers. A common complaint among clients of software engineering projects is a lack of communication. [36] [21] Keeping the client up-to-date regarding project status, issues currently being addressed, accomplishments, and so forth help establish a positive relationship between the client and the development team. This results in an environment of collegial cooperation. Unfortunately, this is currently more often the exception than the rule. For example, while a project plan is practically a "pro forma" element of any project, many do not include a communications plan. [38] [39]

• Risk management. Risk is frequently viewed as something that just occurs unpredictably, requiring the development team to respond extemporaneously. This is a tactical and not very effective view. Certainly, unforeseeable events may happen that threaten the project and must be dealt with. However, there are several methods that can be employed to pre-emptively reduce the severity of risks early in the project. [21] These include:

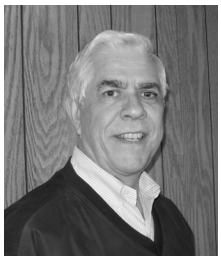
—Identifying factors that could jeopardize the project.
—Assigning a potential monetary cost in case it occurs (or "fires").
—Attaching a probability of occurrence to it.
—Setting aside a contingency fund to address the risk if it fires. [21]
—Mitigating the risk by eliminating what could cause it to "fire."
—Establishing a confidence factor regarding the project's ability to withstand or overcome the risk if it fires. [21]
—Complexity management. Accurately assessing how complex the project is likely to be and working to address challenges early on is another strategic approach to help ensure project success. This process involves assessing complexity via an inventory of complexity factors and challenging each factor to see if it can be mitigated or eliminated altogether. [21] [36] [40] [41]

This more or less holistic approach to the software project management "problem" is a dramatic shift away from the philosophy that technology will "win the day" and toward a philosophy that taps into the incredible potential of the people who are ultimately the foundation of the software engineering profession.

Closing comments

The software engineering profession has tried to develop and use technology as a solution to its project problems for nearly 50 years, but the problems have persisted. Although it may seem a bit risky, it is time for us to try a new approach: focusing on how software projects are managed and, hence, how software engineers are managed. We have little to lose and much to gain.

ABOUT THE AUTHOR



Dr. Lawrence (Larry) Peters has more than 40 years experience in software engineering as a software engineer, instructor, project manager, and consultant. He has worked in the defense, aerospace, telecommunications, and other fields. His area of specialization is software project management. He has published several papers and books focusing on the education, evaluation and importance of the software project manager. He teaches software project management at the M.S. level in Spain via Skype and on-site short courses on software project management.

ljpeters42@gmail.com

REFERENCES

1. Naur, P. and Randell, B. (Editors). (1969, January.) Software Engineering: Report on a Conference Sponsored by the NATO Science Committee. Garmisch, Germany, 7 to 11 October, 1968, Brussels, Scientific Affairs Division, NATO.
2. Rico, D.F. Short History of Software Methods. Downloaded off the web, August 2010. Referenced with author's permission.
3. Peters, L. (1983.) Software Design: Methods and Techniques. Prentice-Hall, Englewood Cliffs, New Jersey.
4. Schlumberger, M. (1991.) Software Engineering Management. Position paper in Proceedings of the 13th International Conference on Software Engineering. 152-153.
5. Boehm, B. (1981.) Software Engineering Economics. Prentice-Hall, Englewood Cliffs, N.J. 486-487.
6. Jensen, R. (2000, August.) Don't Forget About Good Management. CrossTalk, 30.
7. Weinberg, G. M. (1994.) Quality Software Management, Vol. 3: Congruent Action, New York, N.Y. Dorset House Publishing. 15-16.
8. Gulla, J. (2012, February.) Seven Reasons IT Projects Fail. IBM Systems Magazine.
9. Katz, R. (2013.) Motivating Technical Professionals Today. IEEE Engineering Management Review, 2013, 41, (1), 28-37.
10. Bourque, P. & Fairley, R.E. (2014.) Guide to the Software Engineering Body of Knowledge. Version 3.0, IEEE Computer Society. Retrieved from www.swebok.org.
11. Moreno, A., Sanchez-Segura, M-I, Medina-Dominguez, F., Peters, L. & Araujo, J. (2016, May.) Enriching traditional Software Engineering curricula with Software Project Management Knowledge. International Conference on Software Engineering Education and Training (CSEET), Austin, Texas.
12. Project Management Institute, PMBOK®. (2013.) A Guide to the Project Management Body of Knowledge. Fifth Edition. Project Management Institute, Newtown Square, Pennsylvania.
13. Project Management Institute. (2013.) Software Extension to the PMBOK Guide Fifth Edition. Project Management Institute, Newtown Square, Pennsylvania.
14. Peters, L. & Moreno, A. (2015, May.) Educating Software Engineering Managers Revisited – What Software Project Managers Need to Know Today. International Conference on Software Engineering Education and Training (CSEET). Florence, Italy.
15. Cusumano, M. (2004.) The Business of Software: What Every Manager, Programmer, and Entrepreneur Must Know to Thrive and Survive in Good Times and Bad. Free Press, New York, N.Y.
16. Townsend, R. (1970.) Up the Organization: How to Stop the Corporation from Stifling People and Strangling Profits, (commemorative edition). Knopf, New York, N.Y.
17. Chen, J. & Lin, L. (2004). Modeling Team Member Characteristics for the Formation of a Multifunctional Team in Concurrent Engineering. IEEE Transactions on Engineering Management, 15 (2), 111-124.
18. Townsend, R. (1984.) Further Up the Organization: How to Stop the Corporation from Stifling People and Strangling Profits, (commemorative edition). Knopf, New York, N.Y.
19. Peters, L. (2008, May.) Getting Results from Software Development Teams. Microsoft Press Best Practices Series. Redmond, Washington.
20. Kerzner, H. (2013.) Project Management: A Systems Approach to Planning, Scheduling, and Controlling (11th ed.). New York, NY: John Wiley & Sons.
21. Peters, L. (2015, May.) Managing Software Projects On the Edge of Chaos – From Antipatterns to Success. Software Consultants International Ltd., Auburn, Washington. Kindle eBook.
22. Tarim, T. (2012, September.) Making a Transition from Technical Professional to …. IEEE Engineering Management Review, Vol. 10, No. 3, 3-4.
23. Silva, P., Moreno, A. & Peters, L. Software Project Management: Learning from Our Mistakes. IEEE Software, Vol. 32, Issue 3, 40-43.
24. Crosby, P. (1980.) Quality is Free. Signet, New York, N.Y.
25. Petre, M. & Damian, D. (2014, November 14, 16–21.) Methodology and Culture: Drivers of Mediocrity in Software Engineering? Foundations of Software Engineering Conference. Hong Kong, China, 829-832.
26. Crosby, P. (1995.) Quality is still free. McGraw-Hill, New York, N.Y.
27. Thamhain, H. Team Leadership Effectiveness in Technology-Based Project Environments. IEEE Engineering Management Review, 36(1), 165-180.
28. Herzberg, F. (1966.) Work and the Nature of Man. The World Publishing Company, Cleveland, Ohio.
29. Maslow, A.H. (1971.) The Farther Reaches of Human Nature. Viking Press, New York, N.Y.
30. McClelland, D.C. (1961.) The Achieving Society. Van Nostrand-Rheinhold, Princeton, N.J.
31. Grant, A. & Gino, F. A Little Thanks Goes a Long Way: Explaining Why Gratitude Expressions Motivate Prosocial Behavior. Journal of Personality and Social Psychology 2010, Vol. 98, No. 6, 946-955.
32. McAfee, A. (2010, January 7.) The Future of Decision Making: Less Intuition, More Evidence. Harvard Business Review.
33. National Study. (1993.) National Study of the Changing Workforce. Published by the Families and Work Institute, N.Y., N.Y.
34. Myers, C., Staats, B. & Gino, F. (2014, April 18.) 'My Bad!' How Internal Attribution and Ambiguity of Responsibility Affect Learning from Failure. Harvard Business School Working Paper, 14-104.
35. McConnell, S. (2000, July/August.) The Software Manager's Toolkit. IEEE Software.
36. Fleming, Q. & Koppelman, J. (2010.) Earned Value Project Management – Fourth Edition. Project Management Institute. Newtown Square, Pennsylvania.
37. Lipke, W. (2012, January.) Earned Schedule. Retrieved from lulu.com.
38. Belbin, R. (1996.) Management Teams – Why They Succeed or Fail. Butterworth Heineman, London.
39. Dow, W. & Taylor, B. (2008.) Project Communications Bible. Wiley, New York, N.Y.
40. Maylor, H.R., Turner, N. W., & Murray-Webster, R. (2013, July/August.) How Hard Can It Be? Research-Technology-Management, 45-51.
41. Chipulu, M., Neoh, J., Ojiako, U. & Williams, T. (2013.) A Multidimensional Analysis of Project Manager Competences. IEEE Transactions on Engineering Management, Vol. 60, No. 7, 506-517.