Software Project Management Fallacies

Ana M. Moreno[1] and Lawrence Peters[1, 2]
[1]*School of Computer Science, Technical University of Madrid, Boadilla del Monte, Spain*
[2]*Software Consultants International, Auburn, Washington, U.S.A.*

Keywords: Software Project Management, Software Engineering.

Abstract: Software project management plays a critical role in software projects. Therefore, software project management actions have an important impact on software projects and organizations. However, software engineers often become software project managers with little or no training in project management. As a result, sometimes they have to rely on hearsay or their own assumptions to formulate strategies and a plan of action for managing software projects. This has led to several software project management misconceptions or fallacies that can have important negative effects on software projects. This paper examines some relevant fallacies based, on the authors' experience and discusses published material which refutes them. This work contributes to the practice of Software Project Management by identifying and correcting practices which can reduce the success rate of software projects.

1 INTRODUCTION

For a long time, the software project manager has been identified as an important factor in the success of software projects (Boehm, 1981). For example, in (Cone, 1998) the author showed that up to 60% of the cost of software projects can be attributed to personnel turnover and the number one reason why people leave a software organization is their manager. Complementing Boehm's work, Weinberg combined the various factors involved in software projects and found that management was more responsible for success than all other factors combined (Weinberg, 1994). Later, Cusumano made a similar observation stating, "Management, not technology, determines success" (Cusumano, 2004). Although the data that supports these contentions is limited, more recent data confirm that the software project manager is more important to success than all other factors combined (Gulla, 2012).

Today, the advent of the agile philosophy has cast doubt on the value of software project managers. Although it is only a single data point, the experience of Google™ may prove useful (Garvin, 2013) in clarifying the software project manager's role even in agile environments. At one time, the founders of Google subscribed to the thesis that they did not need managers and created a "flat" organization. Details of how to address one issue or another all ended up being sent to the founders of the company, overwhelming their ability to maintain any form of control. Over time, they came to highlight the relevant role that software project managers performed in the company including prioritizing work, helping software engineers to develop their careers, motivating individuals and teams and keeping everyone focused on corporate goals.

So, we must conclude that software project managers are vital to software projects and organizations in many different ways. Therefore, their knowledge, beliefs and actions can affect the outcome of software projects. That is why identifying and refuting mistaken beliefs about software project management (referred to herein as SPM fallacies) is relevant and can have a positive effect on the profession.

There are many definitions for the term, "fallacy". The one used in this paper is misconception or flawed reasoning. Fallacies are not supported by facts or data and the number of beliefs of this type held by software project managers vary from one individual to another and from one organization to the next. The collective set of these beliefs held by some software project managers to be self-evident truths is probably huge. Some of the most relevant according to the authors' experience as software project managers, software consultants and instructors will be examined here. Our aim is to bring SPM fallacies to the attention of

Moreno, A. and Peters, L.
Software Project Management Fallacies.
DOI: 10.5220/0005927001090116
In *Proceedings of the 11th International Joint Conference on Software Technologies (ICSOFT 2016) - Volume 1: ICSOFT-EA*, pages 109-116
ISBN: 978-989-758-194-6

practitioners. This may help to avoid or at least reduce the use of the related practices. It may encourage software project managers to reflect upon other SPM fallacies not discussed in this paper but used in their conduct of software projects.

2 WHERE DID SPM FALLACIES COME FROM?

Fallacies are beliefs that can be the result of misinformation, wishful thinking, ignorance, or hearsay, and as already mentioned, are not based on facts and data. The wishful thinking aspect of fallacies is due to the desire for some simple way to address a complex problem.

Fallacies play a role in software project management due to a lack of sufficient preparation of software project managers for their role in software development (Tomer, 2014). This situation is not totally unexpected since moving from the ranks of software engineer to software project manager subjects the new software project manager to situations they may not have experienced or been trained for. As one moves to a position of authority, one's intuition influences decision making more and more (Taylor, 2011). The higher in the organization one progresses to, the greater the influence of intuition until intuition is dominant.

The relevant role of intuition helps explain why software project managers do not seem to learn from project failures and therefore fallacies are not easy to remove from their body of knowledge. Software project managers' intuition guides their actions and accepting responsibility for a failure reflects on them personally bringing into question their view of their own competence. Instead, the blame is placed on just about any other factor, changing requirements, lack of customer involvement, and so on. Shirking one's responsibility for failure has been shown to inhibit our ability to learn from failure (Myers et al., 2014) and, therefore does not help software project managers to reduce our set of fallacies.

3 SOME RELEVANT SPM FALLACIES

Misguided management practices are not unique to software project management but exist in various forms in all areas of project management. Some are unique to software development due to software engineering's parochial view that software projects are different from all other projects and as such, not subject to the same forces and principles present in other projects (McConnell, 2000).

The situation is somewhat complicated by the fact that not all software project managers hold the same set of beliefs to be self-evident. Additionally, new software project management beliefs may be created with each passing day. Our aim is not to provide a complete list of software project management myths but to discuss and offer data and references to refute some of the most relevant ones according to the authors' professional experience.

Although the different fallacies can be related, we have categorized them into three groups: Team and Productivity, Planning and Scheduling, and Process and Lifecycle.

3.1 Teams and Productivity

In the team and productivity category we group some fallacies that are related to team composition and that can drive decisions that impact on the productivity of such teams.

1. *The Software Project Manager must be the Best Software Engineer on the Team* – This one has been written about and refuted for nearly half a century (Townsend, 1970; Townsend, 1984). There are three major issues which discredit this idea:
   a. The productivity of the team – By putting the most productive software engineer into management, the overall productivity of the team is reduced.
   b. The mentoring capability of the project manager – Unless this is a very special person, they will not be patient with poor performing software engineers and help improve their abilities and mentor them to advance their career.
   c. Effectiveness as a project manager – This high performing person was exceptional because they really liked developing software. Taking them away from what they loved doing results in a disgruntled software project manager who would rather be writing code than dealing with the various aspects of software project management.
2. *The Best Team is Composed of the Best Software Engineers Available* – This concept has been tried in other high technology endeavours and it has consistently failed (Belbin, 1996). This phenomenon even has a name, "The Apollo Syndrome." In part, what happens is that we have

a team of (for lack of a better term) "prima donnas", some of them believing they are smarter and more accomplished than their colleagues. Under these circumstances, they do not work together as a mutually supportive team. Instead, they break up into individual contributors and/or separate factions often competing with each other in order to demonstrate their superiority. The result is detrimental to the project.

3. *Spread Success by Distributing Members of Successful Teams to other Teams* – Empirical and field studies (Staats et al., 2010) have shown that breaking up a team to do this results in the loss of an effective team and the intended spreading of the successful concepts does not happen. If the goal really is to spread successful practices, then we should study what the manager of that project did and emulate it elsewhere. More than this, a study of 1,004 development projects involving more than 11,000 people found that when team familiarity (i.e. team members had worked together before) increased by 50%, defects decreased by 19% and budget deviations decreased by 30% (Huckman and Staats, 2013).
4. *All that I need is One or Two High Performing Software Engineers on the Team to be Successful* –It is often referred to as "stove piping" or the "islands of knowledge" approach. Like the other myths, a software project manager can be successful engaging in it until disaster strikes – the key player leaves the project. While it may be tempting to rely on one or two individuals to "carry" the project it inevitably leads to problems. Successful software development teams take the opposite approach encouraging members to share knowledge in a collaborative, collegial environment (Staats et al., 2011). Often, key players involved in stove piping see retaining vital knowledge as a form of job security. However, these individuals do not realize how much that will restrict their career growth by preventing transfer to a more interesting project. The solution to this situation is to cross-train within the team so that other team members can support the project if a key player becomes unavailable.
5. *We can Multiplex our Best People* – With the increase in work and complexity in so many organizations, it has become common to share highly skilled workers with more than one project. More than 20 years of research on the effects of multitasking on individual productivity have found negative effects on productivity associated with this practice (APA, 2006; Ophira et al, 2009). They found that there is a reduction in productive time of 40% and a greater tendency toward distraction and errors. People are not "plug compatible" and able to instantly switch from one set of project issues to another without some form of "spin up" taking place thereby losing productivity. So, while this practice may continue, the data supporting it are not encouraging.
6. *People Work for Money* – For many years, leading experts on human behaviour have studied and identified why people work (Herzberg, 1966; Maslow, 1971; McClelland, 1961). Although their models differ slightly, one of the things they have in common is that people work for self-esteem, self-realization, and other reasons – not for money. This myth leads software project managers to use money as a means of motivating people to improve their performance. Money, as an incentive for improving productivity has been shown to work for short periods and only in situations involving repetitive activity such as factory work (Ryan and Deci, 2006). But the best way to improve performance is to thank people for their work (Grant and Gino, 2010). That seems paradoxical in that the most costly reward is less effective than one that is free. The thank you does not have to be some sort of public ceremony; a private, one-on-one meeting is all that is required. Some software project managers do not see the need for saying thank you because (in their words), "I do not need to thank them for doing their jobs." Given what we now know, that position will not motivate people to perform at a high level
7. *Offer a Big Reward to get Higher Productivity* – People do not work for money but if the reward is big enough, they will cheat in order to get the reward (Gino and Ariely, 2011). Big rewards (e.g. a trip to some place special, paid for by the company) can undermine ethical behaviour by one or more team members resulting in friction within the team and loss of productivity.
8. *Start with a bigger team* – Some software project managers believe they should start with a larger team than they really need. They believe this strategy (though more costly) will prevent them from getting behind schedule. Empirical and field studies (Staats et al., 2014) have shown that larger teams are less effective than "right sized" teams. Overall, larger teams have been shown to be less efficient and productive.
9. *Putting Pressure on the Team will Improve the Team's Performance* – This one seems reasonable, implying there will be negative consequences to the team if the team is not

successful. The theory is that this pressure should get everyone to make an extra effort. We frequently see this strategy in books and movies. In fact, people work at lower productivity levels when they are under stress (Gardner, 2012). Also, if there is enough pressure placed on the team, the collective knowledge and experience of the team will suffer with team members working independently and not relying on other team members to help them with particularly troublesome software problems or helping other team members with theirs (Gardner, 2012). This results in the collective knowledge of the team being lost (Gardner, 2012). How much pressure is too much pressure? That is one of the "soft" issues successful software project managers have mastered. By maintaining open communications with the team, an effective software project manager can tell when the team is not performing at its highest level and work to reduce the pressure in some way.

10. *Set Challenging Goals, if People Have a Target, They will work to achieve It* – As with some of the other myths listed here, this one is a question of degree, not an absolute. Senior managers often like to call the goals that are set for an organization, "stretch goals." This name implies what the senior manager wants to achieve. That is, to get the team to push themselves to increase or "stretch" their performance. As well intended as this concept may be, studies have shown that if the team perceives the goal as being unachievable, productivity suffers (Ordonez et al., 2009). So the admonition here is to set goals that are achievable. Finding out what the team thinks is achievable must be part of the software project manager's communication skill.

3.2 Process and Lifecycle

These fallacies impact different aspects of the software process, its use, how to improve it, a3s well as the particular software lifecycle to be used in the project.

1. *Requirements Changes Cause Software Project Failures* – Requirements changes are almost inevitable in nearly all software and other technology related projects because requirements definition is a "discovery" process. The project begins with all the stakeholders possessing some vision of what will be produced. Over time, it becomes obvious to some that what they thought was going to be delivered differs significantly from what is likely to be delivered. This results in requirements changes. In the construction industry, requirements changes are a way of life (Peters, 2015; Touran, 2003). The changes themselves in any industry are usually not the problem. Not planning for and accommodating changes in a cost effective manner is. Assuming there will be no changes in requirements and other simplifying assumptions put the project at risk. Changes can cause rework increasing cost and lengthening schedule which can put the project at risk if the process we use does not allow us to accommodate them in a cost effective softy way.
2. *If It isn't Broken, don't Fix It* – This one is popular in many industries but, even though it sounds reasonable, it sets up a culture of maintaining the *status quo*. A better approach is to proactively establish a culture of continual improvement (Weick, 1987). A slight revision of this philosophy to a proactive one results in the revised statement, "If it isn't breaking, don't fix it." This imparts a philosophy of continually monitoring and improving processes, methods, techniques and related matters to do better as a team going forward. If the change does not improve matters, undo it and try something else. The concept here is to constantly monitor what is happening in our project(s) looking for ways to fine tune our methods to reduce cost, improve quality and increase our proficiency. Based on studies of how software engineers view their work and relationship to management (Katz, 2013), such an environment motivates software engineers to higher levels of productivity.
3. *Software Development Processes are Great but if we are Behind Schedule, We won't have Time to use Them* – There have been a lot of arguments against employing any defined development process including that such processes are restrictive and inhibit creativity and innovation. Regarding software development processes, remember, they did not get created in a vacuum but, in general, were the result of work by previous and possibly current software engineering teams focused on improving productivity, improving product quality and so forth. The real test of a software process's value is whether or not it can be used in an emergency (Rombach, 2003).
4. *Not using the Waterfall Lifecycle Improves a Project's Chances of Success* – Anecdotally, this lifecycle model has been taken to task for not being viable in today's software engineering environment. Agile and other approaches have been anecdotally described as being an

improvement without overwhelming evidence. What is needed here are a set of facts and data. Until a study is performed looking at hundreds or thousands of software projects, categorizing their lifecycle models and clearly demonstrating this one is actually true, it must remain a myth.

5. *Technology is the Key to Success in Software Projects* – A study by IBM found that technology isn't the key to success (Gulla, 2012). After categorizing the documented causes of software project failures, that study found that 54% of the failures were attributable to poor project management while only 3% of the failures could be attributed to technical challenges. Dependence on technology tends to spill over into the area of project management with potentially disastrous results. A four year study of 72 multinational product development projects found that project problems that appeared to be technology related actually had, at their source, social, psychological and organizational issues. In retrospect, this seems reasonable since it is people who do the work. To compound matters further for high technology projects, project managers overwhelmingly agree that personnel problems are their most difficult problems to deal with (Maylor et al., 2013; Katz, 2013) and are the ones they are the least trained for.

3.3 Planning and Scheduling

Under the planning and scheduling category we group fallacies that impact estimating and project planing.

1. *Planning and Scheduling are the Same* – Although we can find some published opinions supporting this believe (McConnell, 2000), planning and scheduling are related to be sure but not the same (Kerzner, 2013). A plan is simply a list of tasks and subtasks that must be successfully completed for the project to be deemed a success. A schedule time orders the tasks indicating which must be done first, second and so on as well as which can be done in parallel. These two aspects of the project (the plan and the schedule) are, in a sense, linked by the assignment of specific individuals to tasks. Changes to the plan often result in changes to the schedule, and changes in the schedule can also result in changes in the list of tasks, resulting in an interplay between the two.
2. *Coming Up with the Right Plan Helps Ensure Success* – This is another wishful thinking or "magic bullet" concept. The problem this fallacy creates is that it implies that once the plan (with its corresponding schedule) has been created, the software project manager is done. However, planning and scheduling are continuous activities (Peters and Moreno, 2014), the initial plan is only the start. As General Dwight Eisenhower was quoted as saying, "Plans are nothing, planning is everything". Unforeseen problems will occur that require a change in the project plan and schedule. As the Greek philosopher Heraclitus was quoted as saying, "Change is the only constant."
3. *If Our Project Goes Over Budget or gets Behind Schedule Early on, We can work Harder and Eventually Finish on Budget and on Schedule* – There are, literally, no facts and data to support this one. What facts and data that do exist, based on a study of over 700 projects indicate is that if the project is 15% complete and over budget, its chances of recovering and finishing within its budget are nil (Fleming and Koppelman, 2010). This emphasizes the need for the software project manager to closely monitor cost and schedule right from the start of the project and being willing to take remedial action if the project begins to depart from the plan and schedule.
4. *If We had Better Estimating Methods, We would come Closer to meeting Budget Requirements* – It turns out that no matter how hard we try to accurately estimate any project, we are unknowingly placed at a disadvantage. The problem lies, not in our methods but in ourselves. Research into how people estimate found that we are overly optimistic about our abilities. This human trait is present no matter what formulated method we use (Lovallo and Kahneman, 2003). More recently, a method has become available that helps us back out the effects of over optimism and bound our estimate (Peters, 2015; Flyvberg, 2006). It works so well that the American Planning Association has advised its members to never use traditional estimating methods without also using this method called, "Reference Class Forecasting" (Flyvberg, 2006). What it provides is an estimate plus a set aside or contingency amount which is based on the desired confidence level for the estimate.
5. *Software Engineering is Unique in that Budgets and Schedules are Rarely Met* –Although with some exceptions, this is particularly true when attempting to build something that has never been attempted before. For example, even though man has been building roads and bridges that are seemingly unchallenging tasks for more than two thousand years, overruns in budget and schedule occur today with a great deal of regularity (Flyvberg, 2006). Given all that we should have

learned over the centuries, this seems puzzling until we become cognizant of the 2002 Nobel Prize in Economics winning work of Kahneman who explained this phenomenon of inaccurately estimating as a common trait shared by all human beings (Kahneman, 1994).

4 REMOVING FALLACIES FROM THE MANAGERS' LEXICON

There is no an easy solution for dispatching these and other myths about software project management. Information and education can be the keywords for this challenge, and they should be considered at different levels.

On one hand, at the software project managers level, this paper contains several references which can serve as resources to aid software project managers. But there are other resources. For example, although they do not always abound, software project management books and journal papers and conferences with empirical data about software project management practices, are especially interesting in order to support or refute a particular software project management belief.

Specific knowledge related to software project management has also been recently incorporated into the Software Extension to the Project Management Body of Knowledge (PMBOK, 2013). This Software Extension supplements the PMBOK Guide with specific knowledge related to software projects. The application of this knowledge in real situations can also contribute to reflect about some of our software project management beliefs.

In addition to the referential resources, the experience of instructors is crucial to provide different kinds of empirical knowledge to the audience, as well as a very practical education, in line with what it suggests in general software engineering education. In this sense assistance from accomplished software project managers from industry could help fill this need.

In this educational process about software project management we have to keep in mind that managing a software project involves dealing with issues that frequently do not have a clear or verifiable "right" answer. Examples of these include organizational behaviour, risk management, complexity management, accounting basics, planning, scheduling, estimating and others which have been described as being "wicked" problems (Peters, 2015; Peters, 2008). The term "wicked" is meant to indicate they do not have right or wrong answers and, based on the authors' industrial experience, seem to change continuously. This is clearly a drawback which we face in our profession, however it should not limit our efforts in improving our knowledge.

Finally, notice that, this educational process should be accompanied by other personal lessons that help us to learn to accept our failures and improve (as a popular proverb states "sometimes we win and sometimes we learn").

On the other hand, although educating current and future software project managers is important, their effectiveness in managing projects can be undermined if software engineers and other team members are not also educated regarding the important role the software project manager plays in software projects. This complementary education can help to avoid situations in which experiencing the seemingly mindless actions of software project managers, some software engineers may conclude that software project managers are really not up to the task. That knowledge about software project managers skills can also serve to improve communications between the software project manager and the rest of the team with incumbent improvement in project success. It may also help to prevent some software engineers from becoming software project managers for the wrong reasons (e.g. more money, prestige, better perquisites). The ones that do will at least know in advance just what they are getting into.

The educational process discussed above needs to be complemented from an organizational level. Google's experience in trying to remove software project managers from the organization (Garvin, 2013) provides meaningful evidence about the recognition of the role of software project managers in an organization. The fact that most companies, even major corporations, do not have a clearly defined path to management highlights this lack of understanding (Maylor et al., 2013). In this sense, it is also crucial that the software community recognizes the crucial role of software project management and elevate the status and importance of software project management to being a vital part of the software engineering profession.

5 CLOSING COMMENTS

For half a century, software engineering has focused on technology development to solve software engineering problems. This has produced some

improvements in quality and a linear increase in productivity of approximately one source line per programmer month per year from 1960 to 1990 (Jensen, 2000). Presumably, that linear increase has continued. But software projects continue to fail (Standish Group, 2015).

It has been shown that the management of software projects is where we can obtain the highest return on investment if only we turn our attention to it (Gulla, 2012). This is why we have focussed on discussing misconceptions related to software project management that according to the authors' professional experience are relevant.

We have organized these fallacies into three groups (Team and Productivity, Schedule and Planning, and Process and Lifecycle), with the most numerous ones being related to team management. This is not surprising due to the key role of people in the software development process.

Recognizing the value that competent software project management provides to the project and organization overall is a prerequisite for properly setting the working conditions for software project managers.

An open question arising from this work is: Does removing the mentioned fallacies from the belief systems of software project managers, make them "good" managers?

What a good manager is has been known for at least half a century since Peter Drucker (Drucker, 2006). An exemplary work was done at Google cataloguing the most effective practices (Garvin, 2013). In sum, a "good" manager is someone who:

1. Is a good coach
2. Empowers the team and does not micromanage
3. Expresses interest in and concern for team members' success and personal well-being
4. Is productive and results oriented
5. Is a good communicator – listens and shares information
6. Helps team members with career development
7. Has a clear vision and strategy for the team
8. Has key technical skills that help him or her to advise the team

So, the answer to the previous question should be no. Garvin's list could help software project managers to be especially cautious with the beliefs they have and to confirm them with facts and data to determine whether or not they are sound management practices.

REFERENCES

APA, 2006. American Psychological Association, *Multitasking: Switching costs*. Research in Action, 0 March, 2006.

Belbin, R., 1996. Management Teams – Why They Succeed or Fail, *Butterworth Heineman*, London.

Boehm, B., 1981. *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, N.J., pp. 486-487.

Cone, E., 1998. Managing that Churning Sensation, *Information Week*, No. 680, (May, 1998), 50-67.

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, New York.

Drucker, P., 2006. *Managing for Results* (reissue – originally published 1954), Harper Business, New York, NY.

Fleming, Q. Koppelman, J., 2010. Earned Value Project Management – Fourth Edition, *Project Management Institute*, Newtown Square, PA.

Flyvberg, B., 2006. From Nobel Prize to Project Management: Getting Risks Right, *Project Management Journal*, (August, 2006), pp. 5 – 15.

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

Garvin, D., 2013. How Google Sold Its Engineers on Management, *Harvard Business Review*. https://hbr.org/2013/12/how-google-sold-its-engineers-on-management.

Gino, F., Ariely, D., 2011. The Dark Side of Creativity: Original Thinkers Can be More Dishonest, *Journal of Personality and Social Psychology*, Vol. 102, No. 3, 445-459.

Grant, A.M., Gino F., 2010, A Little Thanks Goes a Long Way: Explaining Why Gratitude Expressions Motivate Prosocial Behavior, *Journal of Personality and Social Psychology*, June, Vol. 98, No. 6, 946 – 955.

Gulla, J., 2012. Seven Reasons IT Projects Fail, *IBM Systems Magazine*, February.

Herzberg, F., 1966. Work and the Nature of Man, *The World Publishing Company*, Cleveland, Ohio.

Huckman, R., Staats, B., 2013. The Hidden Benefits of Keeping Teams Intact, *Harvard Business Review*, (December).

Jensen, R., 2000, Don't Forget About Good Management, *CrossTalk Magazine,* 30.

Kahneman, D., 1994. New Challenges to the Rationality Assumption, *Journal of Institutional and Theoretical Economics*, 150, 18-36.

Katz, R., 2013 Motivating Technical Professionals Today, *IEEE Engineering Management Review*, Volume 41, Number 1, 28-37.

Kerzner, H. R., 2013. *Project Management: A Systems Approach to Planning, Scheduling and Controlling, Wiley Publishing.*

Lovallo, D., Kahneman, D,. 2003. Delusions of Success: How Optimism Undermines Executives' Decisions,

Harvard Business Review, 56-63.
Maslow, A.H., 1971, The Farther Reaches of Human Nature, *Viking Press*, New York, NY.
Maylor, H., Turner, N. and Murray-Webster, R., 2013. How Hard Can It Be? *Research-Technology Management*, (July-August), 46-51.
McClelland, D.C., 1961. The Achieving Society, *Van Nostrand-Rheinhold*, Princeton, NJ.
McConnell, S., 2000. The Software Manager's Toolkit, *IEEE Software*, (July/August, 2013).
Myers, C., Staats, B. and Gino, F., 2014. My Bad! How Internal Attribution and Ambiguity of Responsibility Affect Learning from Failure, *Harvard Business School*, Working Paper 14-104.
Ophira, E., Nass, C., and Wagner, A. 2009. Cognitive Control in Media Multitaskers, *Proceedings of the National Academy of Sciences*, Vol. 106, No. 33.
Ordonez, L., Schweitzer, M., Galinsky, A. and Bazerman, M., 2009. Goals Gone Wild: The Systematic Side Effects of Overprescribing Goal Setting, *The Academy of Management Perspectives*, February, Vol. 23, No. 1, 6-16.
Peters, L. 2008. Getting Results from Software Development Teams, *Microsoft Press Best Practices Series*, Redmond, WA.
Peters, L., Moreno, A, 2014. Educating Software Project Managers – Revisited, *Conference on Software Engineering Education and Training*, Florence, Italy.
Peters, L., 2015. Managing Software Projects on the Edge of Chaos – From Antipatterns to Success, *Software Consultants International Ltd*., Auburn, WA.
Rombach, D. 2003. Teaching How to Engineer Software, keynote address at the *Conference on Software Engineering and Education,* Madrid, Spain, March.
Ryan, R. M., Deci, E. L., 2000. Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions, *Contemporary Educational Psychology*, Volume 25, 54 – 67.
Software Extension to the PMBOK Guide Fifth Edition. 2013. *IEEE Computer Society*.
Staats, B.R., Brunner, D.J., Upton, D.M., 2011. Lean Principles, Learning, and Knowledge Work: Evidence from a Software Services Provider, *Journal of Operations Management*, July.
Staats, B.R., Milkman, K.L. and Fox, C. R., 2014. The Team Sizing Fallacy: Underestimating The Declining Efficiency of Larger Teams, *Forthcoming article in Organizational Behavior and Human Decision Processes*.
Staats, B.R., Gino, F., Pisano, G.P., 2010. Varied Experience, Team Familiarity, and Learning: The Mediating Role of Psychological Safety, Working Paper 10-016, (2010), *Harvard Business School*.
Standish Group, 2015. The Standish Group. The CHAOS Report, West Yarmouth, MA, 2015.Taylor, B., 2011, Why Do Smart People Do Such Dumb Things? *Harvard Business Review,* (January 11, 2011).
Tomer, A., 2014. Software Manageeringment Teaching Project from Software Engineer Perspective, Global Engineering Education Conference (EDUCON, 2014).
Touran, A., 2003. Calculation of Contingency in Construction Projects, *IEEE Transactions on Engineering Management* 50(2), (May, 2003), 135-140.
Townsend, R., 1970. Up the Organization – How to Stop The Corporation from Stifling People and Strangling Profits, *Alfred Knopf*, New York, N.Y.
Townsend, R., 1984. Further Up the Organization – How to Stop Management from Stifling People and Strangling Profits, *Alfred Knopf*, New York, N.Y.
Weick, K., 1987. Organizational Culture as a Source of High Reliability, *California Management Review*, 29/2, 112-127.
Weinberg, G., 1994. *Quality Software Management, vol. 3: Congruent Action*, New York, N.Y.: Dorset House Publishing, 15-16.