Association for Information Systems
AIS Electronic Library (AISeL)

International Research Workshop on IT Project Management 2013

International Research Workshop on IT Project Management (IRWITPM)

12-14-2013

The Planning Fallacy as an Explanation for Over-Requirement in Software Development

Ofira Shmueli
Ben-Gurion University of the Negev, ofirash@post.bgu.ac.il

Nava Pliskin
Ben-Gurion University of the Negev, pliskinn@bgu.ac.il

Lior Fink
Ben-Gurion University of the Negev, finkl@bgu.ac.il

The Planning Fallacy as an Explanation for Over-Requirement in Software Development

Ofira Shmueli
Ben-Gurion University of the Negev
ofirash@post.bgu.ac.il

Nava Pliskin
Ben-Gurion University of the Negev
pliskinn@bgu.ac.il

Lior Fink
Ben-Gurion University of the Negev
finkl@bgu.ac.il

ABSTRACT

Over-Requirement occurs in software development projects when a software product is specified beyond the actual needs. This study shows empirically that Over-Requirement happens partially due to the Planning Fallacy, i.e., the tendency of people to underestimate the time needed to complete a task. Underestimating the time needed to develop a software feature during project planning, we argue, may lead to including within the project scope more required and unrequired features than can be completed by the project deadline. To investigate this argument, we conducted an experiment in which participants were asked to estimate the time it would take to develop various software features in a software development project and then, given the project's duration, to recommend which of the features to include within scope. The results confirmed that the Planning Fallacy occurs in the context of software development and influences the Over-Requirement phenomenon.

Keywords

Planning fallacy, software development, over-requirement, over-specification, gold-plating, experiment.

INTRODUCTION

The phenomenon of Over-Requirement, also termed over-specification or gold-plating, occurs when a product or a service is specified beyond the actual needs of the customer or the market (Boehm and Papaccio, 1988; Ronen and Pass, 2008). In software development projects, Over-Requirement is a major risk since, once a development project is launched, it is very difficult to cut features off its scope, even when some features are not really necessary (Dominus, 2006). Despite its negative impacts, Over-Requirement is a common phenomenon reported to pertain to at least 30% of developed features (Coman and Ronen, 2009).

The purpose of this work is to show that Over-Requirement is partially due to the Planning Fallacy which, according to Kahneman and Tversky (1979), refers to the tendency of people to underestimate the time needed to complete a task even in the presence of knowledge about past failures and time overruns. However, less underestimation of task completion times has been found in situations in which past experiences are perceived as relevant to the task at hand (Griffin and Buehler, 2005) and in the case of predictions performed by an uninvolved observer (Buehler, Griffin and Ross, 1995; Buehler, Griffin, Lam and Deslauriers, 2012). The Planning Fallacy has been demonstrated in a wide variety of tasks, including origami folding, school work, and computer programming (Connolly and Dean, 1997; Pezzo, Litman and Pezzo, 2006).

We argue that in the case of software development, such underestimation of task completion times at the stage of project scoping may lead to including in the project more features (some of which are Over-Required) than can be completed within time. To investigate this argument, we conducted an experiment that required participants, referring to a specific to-be-developed software project, to (a) estimate development times of various software features and (b) recommend which of the features to include in the project scope given the duration of the project. The experiment results confirmed that the Planning Fallacy occurs in the context of software development and influences the extent of Over-Requirement. Moreover, for software engineers in both development and consulting roles, knowing actual previous development times may improve time estimations and decrease the magnitude of Over-Requirement. Yet, this knowledge eliminates neither the Planning Fallacy nor the inclusion of Over-Required features. In addition, whether they have knowledge of previous development times or not, software developers tend

to include in scope more features than software consultants, including Over-Required features. Finally, testing the relation between time estimation and Over-Requirement reveals that the more underestimation of development time exists, the more Over-Requirement occurs.

The main contribution of this work to both research and practice is in improving the understanding of Over-Requirement which, despite its negative influence, has barely been explored. This research proposes and empirically supports a behavioral explanation for the inclusion of Over-Required features in project scope, innovating in its approach of exploring behavioral effects in the context of software development as well as exploring their consequences on the whole software development project.

The next section of this paper presents the theoretical background for the experiment as well as this study's hypotheses. Then the following sections detail the experiment method and results. Finally, the paper concludes with a discussion of the results as well as this work's contributions and limitations.

THEORETICAL BACKGROUND AND HYPOTHESES

Over-Requirement

Boehm (1991) included Over-Requirement among the top 10 software development risks, and NASA (1992) listed Over-Requirement among the eight "don't do" warnings in the software-development context. Similarly, Over-Requirement was mentioned as one of the top 10 or 20 risks in most studies devoted to identifying, classifying, and ranking software-development risks (Baccarini, Salm and Love, 2004; Houston, Mackulak and Collofello, 2001; Khanfar, Elzamly, Al-Ahmad, El-Qawasmeh, Alsamara and Abuleil, 2008; Schmidt, Lyytinen, Keil and Cule, 2001). The major damages that make Over-Requirement risky vary from delayed launch, through excessive complexity, to demise of an entire company. The list of negative outcomes includes exceeding planned resources as time or budget (Buschmann, 2009; Coman and Ronen, 2009, 2010) and the undesirable impact on software complexity, reliability, and maintainability (Battles, Mark and Ryan, 1996; Buschmann, 2010; Coman and Ronen, 2009, 2010; Elliott, 2007; Westfall, 2005). In addition, project resources are being wasted on functionality of no value, instead of on core-business functionality (Coman and Ronen, 2009, 2010; Elliott, 2007; Westfall, 2005), and poor outcomes may affect the users and the entire organization (Coman and Ronen, 2009, 2010; Kautz, 2009; Rust, Thompson and Hamilton, 2006). Nevertheless, Over-Requirement is a common, hardly-reversible phenomenon since excessive extra features introduced during the requirement-engineering phase or later are very rarely cut off scope (Dominus, 2006; Wetherbe, 1991).

Planning Fallacy

The Planning Fallacy refers to the tendency of people to underestimate the time needed to complete a task even in the presence of experience of over-runs in similar tasks. The term for this behavioral effect was first proposed by Kahneman and Tversky (1979), who also acknowledged the generalizability of this effect to both experts and laypersons. Buehler, Griffin and Peetz (2010) emphasized later on its paradoxical nature as manifested by the combination of optimistic views about the future and realism about the past. The Planning Fallacy has been found for a wide variety of desirable and undesirable tasks, including origami folding (Pezzo et al., 2006) as well as school work, tax-form completion, and computer programming (Connolly and Dean, 1997). The Planning Fallacy seems to be a result of motivational factors, but Kahneman and Tversky (1979) suggested that it is also a consequence of the tendency of people to focus on the specific problem rather than on the distribution of outcomes of similar cases. A known psychological observation is that people tend to take credit for positive outcomes but attribute negative outcomes to external factors (Lovallo and Kahneman, 2003), providing another explanation for the tendency of people to provide overly optimistic time estimations. Overall, underestimation of completion times is a negative phenomenon because it does not shorten the actual time the task takes, unless the task is very simple and performed immediately (Buehler, Peetz and Griffin, 2010), negatively affecting the entire project. Underestimation done in early phases of the project tends to persist and leads to a sustained false optimism along later project phases (Kutsch, Maylor, Weyer and Lupson, 2011).

Decomposing a large task into small enough sub-tasks was shown to decrease time underestimation as a result of a segmentation effect (Forsyth and Burt, 2008), whether the task is holiday shopping, document formatting (Kruger and Evans, 2004), or code writing (Connolly and Dean, 1997). The more complicated the task is, the more can underestimation reduction be attained (Kruger and Evans, 2004). Motivating accuracy in time estimation by offering

monetary incentives for accurate time prediction, rather than motivating speed, might help as well (Brunnermeier, Papakonstantinou and Parker, 2008). However, while the Planning Fallacy applies to predictions about one's own tasks, predictions done by an external party are less prone to the Planning Fallacy (Buehler et al., 1995; Buehler et al., 2012).

This work investigates the Planning Fallacy in the context of software-development projects, focusing on the underestimation of development times. Time underestimation is bound to influence project scoping because underestimating the time it would take to develop software features may lead to the inclusion of more features within project scope than actually possible to complete by the project deadline. This scope overload may not only lead to time overruns but also to the Over-Requirement phenomenon, as hypothesized next.

Hypotheses

Experts and laypersons alike were found to exhibit the Planning Fallacy, ignoring knowledge gained in the past about the distribution of outcomes (Kahneman and Tversky, 1979). Even upon increasing the salience of past memories by asking about past experiences immediately before asking for time predictions did not result in less optimistic forecasts (Griffin and Buehler, 2005). Perceived irrelevance of past experiences was demonstrated also in the case of software engineers who, knowing that past projects tended to be late, did not use the results of past projects to inform future predictions (Buehler, Griffin and Peetz, 2010). However, a reduction of the Planning Fallacy was demonstrated under the recall relevance condition, in which subjects not only recall past completion times but also describe how the current task might be similar to past tasks (Griffin and Buehler, 2005). This study aims to find out if the role assumed by a software engineer, whether developer or consultant, makes a difference.

Software development projects are known to suffer from massive time overruns (Standish Group, 2009). The human tendency to underestimate and to produce overly optimistic development schedules was noted as the most common reason for failure of software projects (Charette, 2005; Nelson, 2007). Years of evolutionary improvement of software-development methods and tools, as well as the vast experience gained, have barely improved time estimation, probably due to the perceived irrelevance of the past based on the perception of specific projects as unique (Buehler et al., 2010). Thus, knowledge of the past does not diminish the Planning Fallacy. Yet, for software engineers, the presence of concrete and relevant knowledge concerning development times of similar software tasks in the past, as in the recall relevance condition, may reduce underestimation of the time it would take to complete a software task, while the absence of such knowledge may increase underestimation. Thus, we hypothesize that in the absence of past relevant knowledge, time underestimation increases. Hence,

H1: Software engineers without past relevant knowledge of development times will provide lower total time estimations for software-development tasks than those with past relevant knowledge

The Planning Fallacy and underestimation apply to predictions about one's own tasks. Yet, predictions done by uninvolved observers usually reduce underestimation and sometimes even result in overestimation (Buehler et al., 1995). Moreover, simply imagining the upcoming task from the perspective of another person moderates the optimistic prediction bias and reduces underestimation, helping to generate more realistic predictions by reducing cognitive and motivational processes that typically contribute to bias (Buehler et al., 2012).

In software-development projects, estimations of completion times are done usually by the software developers themselves, estimating the time it would take each of them to develop the software features. Buehler et al. (2010) have shown that, when asked to predict software-development times, developers neglect experience gained in past projects and consider the current project unique. Regarding the software features that developers are going to develop, we argue that they might be more prone to bias in estimating completion times than uninvolved observers, such as consultants. Thus, we hypothesize that the role of the person affects the degree of underestimation. Hence,

H2: Software developers will provide lower total time estimations for their software-development tasks than software consultants

Scope or feature creep in software-development projects is the tendency to include too many features within project scope (Buschmann, 2009, 2010). We argue that underestimating the development times may impact the number of features included in the scope of the project. In other words, underestimating development times may cause the illusion of having more time than there really is, resulting in the inclusion of more features within scope than can be

actually completed on time. Again, having past relevant knowledge may influence the number of features to be included, as is the software engineer's role, whether developer or consultant. Thus, we hypothesize that the number of features included within scope increases in the absence of past relevant knowledge and in the case of software developers, consistent with the two main effects on time estimations described in H1 and H2. Hence,

H3: Software engineers without past relevant knowledge of development times will plan more features to be included within project scope than those with past relevant knowledge

H4: Software developers will plan more features to be included within project scope than software consultants

Over-Requirement in software development projects is the tendency to include unneeded features within project scope. We argue that H3 and H4 can be extended beyond the number of features included within scope also to the number of Over-Required features. This argument is based on the assumption that the underestimation of development times and the inclusion of more features within project scope are not contingent on feature importance. Again, past relevant knowledge and the individual perspective may impact the magnitude of Over-Requirement. Thus, we hypothesize that the number of Over-Required features included within scope increases in the absence of past relevant knowledge and in the case of software developers. Hence,

H5: Software engineers without past relevant knowledge of development times will plan more Over-Required features to be included within project scope than those with past relevant knowledge

H_6: Software developers will plan more Over-Required features to be included within project scope than software consultants

Finally, we wish to draw a direct line between time underestimation and Over-Requirement. We wish to test the prediction that the underestimation of development times leads people to not only overestimate how much they can accomplish in a given period of time (Nouri, Jamali and Ghasemi, 2012) but also to include more Over-Required features within project scope. In other words, regardless of their past relevant knowledge and their role perspective, underestimating development times by software engineers increases the number of Over-Required features they include within scope. Hence,

H7: The more software engineers underestimate total development time the more Over-Required features they will plan to be included within project scope

The research model depicting the above hypotheses is presented in Figure 1.

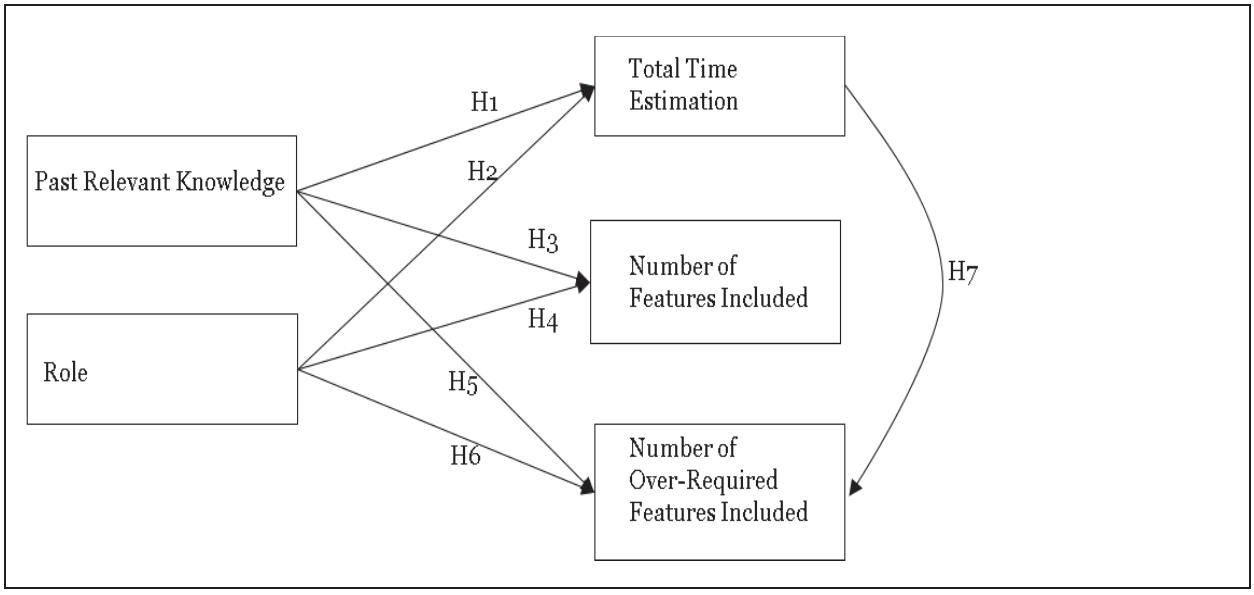

Figure 1. Research Model

METHOD

An experiment was conducted to empirically test the research model depicted in Figure 1 and the seven hypotheses presented above. The experiment targeted advanced undergraduate Industrial Engineering & Management students majoring in the Information Systems while they took in the junior year the compulsory course "Automation and Computer-integrated Manufacturing". By the time that this experiment took place they already experienced performing tasks similar to the task demanded of them in the experiment as part of their duties in this course and previous courses. As an incentive to participate in the experiment, each participant received an extra bonus point (1 out of 100) in the course grade. It is worth noting that engineering undergraduate students in Israel, where the experiment took place, are older than their typical counterparts elsewhere in the world and most of them work part time while studying. In our experiment, 64% of the participants were 26 years old and above, yet only few had past work experience in industry. Since it is difficult to collect data from participants employed by industry when a controlled environment is required, relying on students as participants is acceptable in experiments exploring the behavior and the decision making of software developers and managers (Andres, 2001; Keil, Im and Mähring, 2007; Keil, Tan, Kwok-Kee, Saarinen and et al, 2000; Umapathy , Purao and Barton , 2008).

A pilot study with a sample of 15 students from the same population as in the experiment was performed to test the experimental design and manipulations. Some minor modifications were made according to feedback received from participants in the pilot study.

The main experiment was based on a factorial design of 2×2, manipulating two dichotomous independent variables: Past Relevant Knowledge (with or without knowledge regarding development times of similar features in the past) and Role (software developer or software consultant). Prior to randomly assigning the target sample to four groups, 10 participants were randomly excluded from the target population and assigned to a reference group that performed a side experiment in a parallel session without any manipulations. All other participants participated in the main experiment and were randomly assigned to one of four groups: developers with past relevant knowledge, developers without past relevant knowledge, consultants with past relevant knowledge, and consultants without past relevant knowledge. In the three-step main experiment, which took about half an hour, the manipulation of Role took place during the first step whereas the manipulation of Past Relevant Knowledge took place in the second step.

In the first step of the experiment, participants were told that they were employees of a big software company (The Company, hereinafter). Then, for manipulating the independent variable Role, half of the participants were randomly assigned to a developer role by being told that they were members of the development team in a development project and half were randomly assigned to a consultant role by being told that they were members of the consulting team in the same project. Then, a fictitious case was presented, according to which The Company is planning a software-development project for operating a robot whose purpose is to use blocks of different sizes, which arrive on a conveyor, to build three towers. Additional details as the sizes and locations of the towers were also provided, along with a list of 16 features of the planned software that the customer mentioned in early meetings with representatives of The Company. The feature list was deliberately composed of features of different importance for the project's goal as determined by two course instructors independently: five features on the list which both instructors considered unnecessary, were categorized as Over-Required, seven features as essential, and four as optional. In the second step of the main experiment, participants in a developer role were asked to estimate how long it would take them to develop each feature while, on the other hand, participants in a consultant role were asked how long it would take the developers to develop each feature. Then, for manipulating the independent variable Past Relevant Knowledge, half of the participants assigned earlier to each of the two roles were further randomly assigned to the group with (without) past knowledge by being informed (or not), for each feature, about the time it took to develop a similar feature in the past. Only in the third step of the main experiment participants were informed that the project duration is planned to be 18 hours. They were then asked to choose features to be included within the project scope, taking into consideration not only their earlier estimations of development times given while unaware of the upper bound on project duration but also the set bound of 18 hours.

For testing the different hypotheses, three dependent variables were calculated based on the answers of each participant in the main experiment: (a) Total Time Estimation, (b) Number of Features Included, and (c) Number of Over-Required Features Included. The Total Time Estimation summed all 16 time estimations for the development of features on the list. The Number of Features Included counted the number of features that the participant included

within project scope out of the full list of 16 features. The Number of Over-Required Features Included counted the number of features that the participant included within project scope out of the list of five features determined as unnecessary by both course instructors.

Participants in the reference group received a questionnaire with a similar introduction, presenting the project goal and the list of features. They were asked to express their opinion whether each feature was essential in one question, and whether each feature was unnecessary in another question. Their answers landed support to the list of unnecessary features, as determined by the course instructors. On average, 7 out of 10 members of the reference group categorized as unnecessary each of the five features determined unnecessary by course instructors, compared to only 1.3 out of 10 members of the reference group who, on average, categorized as unnecessary each of the other 11 features. The feature valuation by the reference group thus confirmed that participants in the four experiment groups, who belonged to the same target population, were by themselves capable of not only figuring out the importance of the features on the list but also of identifying Over-Required features.

RESULTS

Data collection yielded 85 gender-balanced participants, 75 in the main experiment and 10 in the reference group, and data analysis was pursued via analysis of variance (ANOVA) in SPSS. Overall, the mean of Total Time Estimation provided by participants was 23.03 hours (17.73 hours for those without past relevant knowledge and 28.18 hours for those with that knowledge), almost half of the total time of estimations we presented to participants as past relevant knowledge (43.50 hours). Since our experiment procedure uses homogenous groups with randomly assigned participants, these consistent differences across all 16 features provide evidence for treatment effectiveness since other possible explanations are controlled for. Furthermore, the mean of Number of Over-Required Features Included for all participants was 2.13, implying that participants included within scope almost half of the five unnecessary features. These findings served as a preliminary indication that the experiment was able to capture the Planning Fallacy and Over-Requirement in software development.

Table 1 shows the cell sizes for the four experimental conditions. Table 2, Table 3, and Table 4 show the estimated marginal means (and standard errors) of the three dependent variables. As can be seen in these tables, the group of consultants with Past Relevant Knowledge provided the highest mean of Total Time Estimation (30.95 hours), the lowest mean of Number of Features Included (8.42), and the lowest mean of Number of Over-Required Features Included (1.11). In contrast, the group of developers without Past Relevant Knowledge provided the lowest mean of Total Time Estimation (17.66 hours), the highest mean of Number of Features Included (13.28), and the highest mean of Number of Over-Required Features Included (3.61). Overall, the means of Total Time Estimation for developers and consultants without Past Relevant Knowledge were lower than for developers and consultants with Past Relevant Knowledge. At the same time, the Number of Features Included and the Number of Over-Required Features Included were higher. With and without Past Relevant Knowledge, as can be seen in Tables 3 and 4, the means of Number of Features Included and Number of Over-Required Features Included by developers were higher than by consultants.

	Past Relevant Knowledge		
Role	With	Without	Total
Developer	19	18	37
Consultant	19	19	38
Total	38	37	75

Table 1. Cell Sizes of Experimental Conditions

	Past Relevant Knowledge	
Role	With	Without
Developer	25.41 (2.37)	17.66 (3.34)
Consultant	30.95 (4.53)	17.80 (1.65)

Table 2. Marginal Means (Standard Errors) for Total Time Estimation in Hours

Role	Past Relevant Knowledge	
	With	Without
Developer	11.21 (0.72)	13.28 (0.67)
Consultant	8.42 (0.65)	10.79 (0.63)

Table 3. Marginal Means (Standard Errors) for Number of Features Included

Role	Past Relevant Knowledge	
	With	Without
Developer	2.37 (0.40)	3.61 (0.40)
Consultant	1.11 (0.30)	1.53 (0.38)

Table 4. Marginal Means (Standard Errors) for Number of Over-Required Features Included

Table 5 presents the ANOVA results for the full factorial models for the three dependent variables, aimed at testing H1 through H6. As can be seen in the first column of Table 5, there is a main effect of Past Relevant Knowledge on Total Time Estimation (F=10.933, p<0.01), supporting H1, while H2 is not supported because there is no main effect of Role (F=0.805, p=0.373). Testing H3 and H4, concerning the impact of the two independent variables on the Number of Features Included (second column in Table 5), reveals significant effects of both Past Relevant Knowledge (F=10.754, p<0.01) and Role (F=15.225, p<0.001), respectively supporting H3 and H4. The third column in Table 5 presents two more significant main effects, those of Past Relevant Knowledge (F=4.971, p<0.05) and Role (F=20.129, p<0.001) on the Number of Over-Required Features Included, respectively supporting H5 and H6. All the differences between the two conditions of Past Relevant Knowledge (with or without) and the two conditions of Role (developer or consultant) for all three dependent variables were in the hypothesized direction.

	Total Time Estimation	Number of Features Included	Number of Over-Required Features Included
Past Relevant Knowledge	10.933 **	10.754 **	4.971 *
Role	0.805	15.225 ***	20.129 ***
Role × Past Relevant Knowledge	0.728	0.050	1.212
R^2	0.150	0.266	0.267
Adjusted R^2	0.114	0.235	0.236

Table 5. ANOVA Results (F Values are Shown; * p<0.05, ** p<0.01, *** p<0.001)

Figure 2 graphically depicts these differences for the Number of Over-Required Features Included, confirming the main effects on over-requirement. Concerning H7, a Pearson correlation test revealed a significant negative correlation between Total Time Estimation and the Number of Over-Required Features Included (r=-o.398, p<0.01). As hypothesized, therefore, the lower Total Time Estimation is, the more Over-Required features are included within scope.

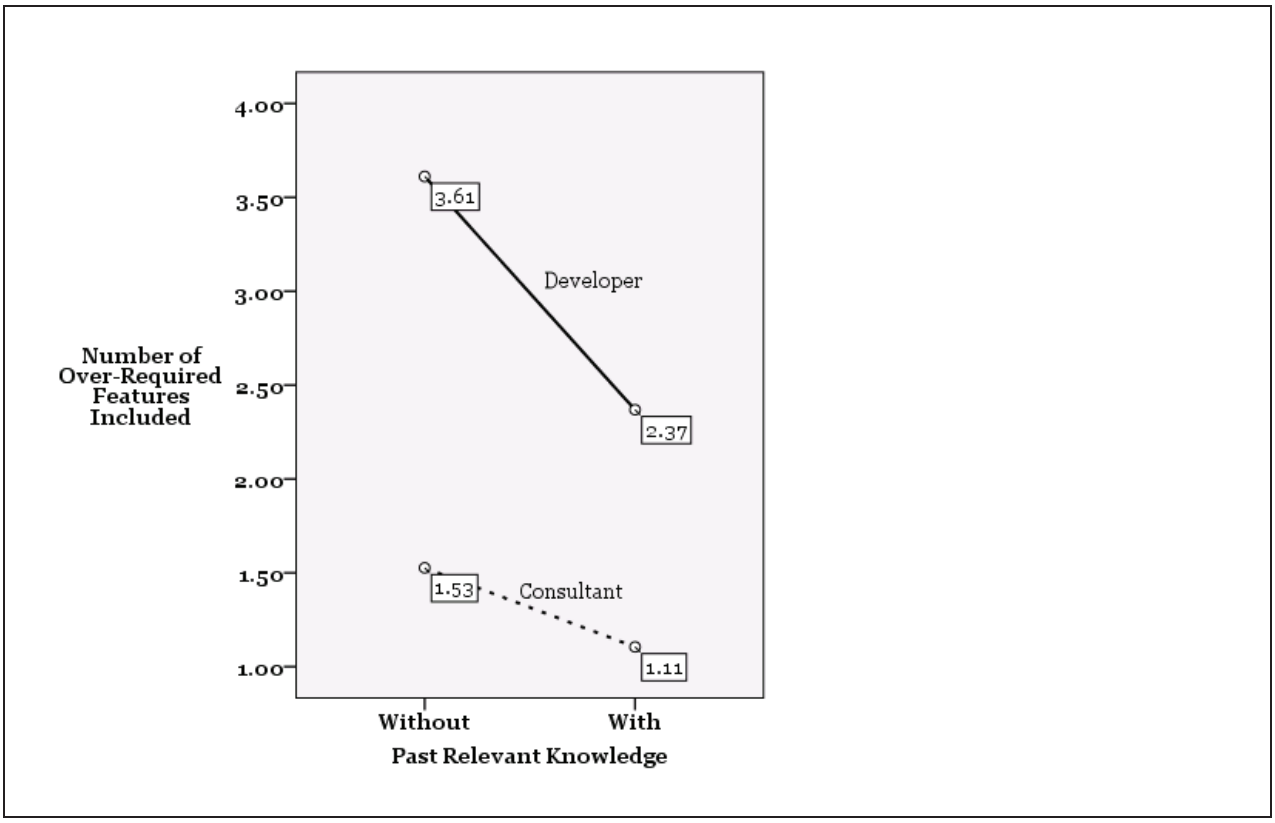

Figure 2. Marginal Means of the Number of Over-Required Features Included

DISCUSSION AND CONCLUSION

The findings of the experiment confirm findings of past studies that the Planning Fallacy plays a role in software development. Moreover, this study finds a relationship between the Planning Fallacy and Over-Requirement. Due to the Planning Fallacy, software developers and consultants alike tend to underestimate the time required to develop software features. Underestimation of software-development time can lead, on one hand, to optimistic time planning for the entire development project and, on the other hand, to Over-Requirement, i.e., planning more features within project scope than can be accomplished by the deadline (Nouri et al., 2012), including Over-Required features.

The effects of past relevant knowledge on the estimation of development time (H1), on the number of features included within project scope (H3), and on the number of Over-Required features included within project scope (H5) have been found significant. As part of their course duties, all participants had the experience of developing similar features. Yet, only half the participants composing two of the four groups were deliberately presented with past development times as relevant to the tasks at hand to ensure that this Past Relevant Knowledge condition holds (Griffin and Buehler, 2005). Having explicit knowledge of past development times seems to reduce time underestimation as well as scope overloading and Over-Requirement. The results thus demonstrate the positive influence of past relevant knowledge and experience on time estimations of software development and on project scoping. The results show, however, that underestimation is reduced but not eliminated by past relevant knowledge.

The role, whether developer or consultant, has been found to have a significant influence on the number of features included within scope, including Over-Required features (H4 and H6 were supported). Yet, there is no significant difference in time estimations provided by developers and consultants (i.e., H2 was not supported). Thus, despite the finding that developers do not provide significantly lower estimations (for tasks to be developed by them) than consultants (for tasks to be developed by others), developers tend to include more features and more Over-Required features within project scope than consultants. These significant differences between developers and consultants may imply other behavioral explanations for the Over-Requirement phenomenon beyond the Planning Fallacy. One

possible explanation might be the endowment effect, due to which ownership feelings that might be elevated when holding a developer's perspective, as opposed to a consultant's perspective, lead to regarding the features under development as more valuable than they really are (Thaler, 1980). Another explanation might be the IKEA effect, due to which feelings of attachment toward the features under development are gained, as a result of cognitive effort invested by developers more intensely than by consultants in analyzing and estimating the time needed to develop each feature (Norton, Mochon and Ariely, 2012). Both explanations rely on the assumption that since consultants are less involved, they are less prone to behavioral biases associated with ownership (endowment) or effort (IKEA). Nevertheless, despite holding a third-person perspective, which the Planning Fallacy regards as less biased subjects, their lesser involvement does not seem to impact the consultants' time estimations. It may very well be that the experimental manipulations failed to cause participants in the consultant role to be completely uninvolved when time estimations were requested since, for the sake of accurate estimations and for the sake of mutual baseline with developers, they were specifically asked to assess the time needed for their "friends in the development team who joined The Company together with them".

Finally, the last hypothesis (H7) was supported, demonstrating the negative relationship between the estimation of development time and the number of Over-Required features included within scope, in the sense that the lower time estimations are, the more Over-Requirement occurs. Since the experiment steps were ordered in a way that subjects were asked to determine the project's scope only *after* they estimated feature completion times, this finding confirms the impact of the Planning Fallacy on Over-Requirement.

The findings of this study are relevant to both practitioners and researchers. Developers and consultants in software development projects should strive, to the extent possible, to base their estimations on the outcomes of past projects of their own and of others. They should resist the inclination to consider each software development project as a unique endeavor and, instead, seek to find relevance in previous projects, which can then serve as a baseline for estimations. In addition, managers of software development projects should be aware that developers are more biased than consultants regarding the inclusion of Over-Required features in scope and that the underestimation of development time is at the root of this phenomenon.

From a research perspective, making the connection between the Planning Fallacy and the phenomenon of Over-Requirement in software development projects opens new directions for future research regarding other behavioral effects that might be relevant to Over-Requirement, such as the endowment and IKEA effects (Norton et al., 2012; Thaler, 1980). Future research might also address the two primary limitations of this study. First, although participants were advanced undergraduate students with the relevant knowledge for the task, most of them did not have real development experience in industry. Second, the experiment took about half an hour and tried to emulate reality concerning time-estimation and project-scoping tasks and the roles of developer and consultant. Yet, it is seldom possible in an experiment to account for such aspects as human interactions, organizational culture, and maturity. To address both limitations, further research can target full-time software developers or software consultants at the phase of project scoping and ask them relevant questions about feature development time and features to be included in scope. Notwithstanding its limitations, this study demonstrates the implications of the Planning Fallacy for software development projects in general and for Over-Requirement in particular. Our findings confirm the notion that the lens of behavioral effects has the potential to advance the understanding of software development processes.

REFERENCES

Andres, H. P. (2001) A contingency approach to software project coordination, *Journal of Management Information Systems,* 18, 3, 41-70.

Baccarini, D., Salm, G. and Love, P. (2004) Management of risks in information technology projects, *Industrial Management and Data Systems,* 104, 4, 286-295.

Battles, B. E., Mark, D. and Ryan, C. (1996) An open letter to CEOs: How otherwise good managers spend too much on information technology, *The McKinsey Quarterly,* 1996, 3, 116-127.

Boehm, B. (1991) Software risk management: Principles and practices, *IEEE Software,* 8, 1, 32-41.

Boehm, B. and Papaccio, P. (1988) Understanding and controlling software costs, *IEEE Transactions on Software Engineering,* 14, 10, 1462-1477.

Brunnermeier, M. K., Papakonstantinou, F. and Parker, J. A. (2008) An economic model of the planning fallacy (Working Paper No. 14228), Princeton University: National Bureau of Economic Research.

Buehler, R., Griffin, D. and Peetz, J. (2010) Chapter one-the planning fallacy: Cognitive, motivational, and social origins, *Advances in Experimental Social Psychology,* 43, 1-62.

Buehler, R., Griffin, D. and Ross, M. (1995) It's about time: Optimistic predictions in work and love, *European Review of Social Psychology,* 6, 1, 1-32.

Buehler, R., Griffin, D., Lam, K. and Deslauriers, J. (2012) Perspectives on prediction: Does third-person imagery improve task completion estimates? *Organizational Behavior and Human Decision Processes,* 117, 1, 138-149.

Buehler, R., Peetz, J. and Griffin, D. (2010) Finishing on time: When do predictions influence completion times? *Organizational Behavior and Human Decision Processes,* 111, 1, 23-32.

Buschmann, F. (2009) Learning from failure, part 1: Scoping and requirements woes, *IEEE Software,* 26, 6, 68-69.

Buschmann, F. (2010) Learning from failure, part 2: Featuritis, performitis, and other diseases, *IEEE Software,* 27, 1, 10-11.

Charette, R. N. (2005) Why software FAILS, *IEEE Spectrum,* 42, 9, 42-49.

Coman, A. and Ronen, B. (2009) Overdosed management: How excess of excellence begets failure, *Human Systems Management,* 28, 3, 93-99.

Coman, A. and Ronen, B. (2010) Icarus' predicament: Managing the pathologies of overspecification and overdesign, *International Journal of Project Management,* 28, 3, 237-244.

Connolly, T. and Dean, D. (1997) Decomposed versus holistic estimates of effort required for software writing tasks, *Management Science,* 43, 7, 1029-1045.

Dominus, M. (2006). *Creeping featurism and the ratchet effect.* Retrieved March 17, 2013, from http://blog.plover.com/prog/featurism.html

Elliott, B. (2007) Anything is possible: Managing feature creep in an innovation rich environment, *Proceedings of the IEEE International Engineering Management Conference,* July 29 - Aug 1, Austin, TX, 304-307.

Forsyth, D. and Burt, C. (2008) Allocating time to future tasks: The effect of task segmentation on planning fallacy bias, *Memory & Cognition,* 36, 4, 791-798.

Griffin, D. and Buehler, R. (2005) Biases and fallacies, memories and predictions: Comment on roy, christenfeld, and McKenzie (2005), *Psychological Bulletin,* 131, 5, 757-760.

Houston, D. X., Mackulak, G. T. and Collofello, J. S. (2001) Stochastic simulation of risk factor potential effects for software development risk management, *The Journal of Systems & Software,* 59, 3, 247-257.

Kahneman, D. and Tversky, A. (1979) Intuitive prediction: Biases and corrective procedures, *TIMS Studies in Management Science,* 12, 313-327.

Kautz, K. (2009) The impact of pricing and opportunistic behavior on information systems development, *Journal of Information Technology Theory and Application,* 10, 3, 24-41.

Keil, M., Im, G. P. and Mähring, M. (2007) Reporting bad news on software projects: The effects of culturally constituted views of face-saving, *Information Systems Journal,* 17, 1, 59-87.

Keil, M., Tan, B. C. Y., Kwok-Kee, W., Saarinen, T. and et al. (2000) A cross-cultural study on escalation of commitment behavior in software projects, *MIS Quarterly,* 24, 2, 299-325.

Khanfar, K., Elzamly, A., Al-Ahmad, W., El-Qawasmeh, E., Alsamara, K. and Abuleil, S. (2008) Managing software project risks with the chi-square technique, *International Management Review,* 4, 2, 18-29.

Kruger, J. and Evans, M. (2004) If you don't want to be late, enumerate: Unpacking reduces the planning fallacy, *Journal of Experimental Social Psychology,* 40, 5, 586-598.

Kutsch, E., Maylor, H., Weyer, B. and Lupson, J. (2011) Performers, trackers, lemmings and the lost: Sustained false optimism in forecasting project outcomes -- evidence from a quasi-experiment, *International Journal of Project Management,* 29, 8, 1070-1081.

Lovallo, D. and Kahneman, D. (2003) Delusions of success: How optimism undermines executives' decisions, *Harvard Business Review,* 81, 7, 56-63.

NASA. (1992) Recommended approach to software development (3rd ed.). Greenbelt, MD: Goddard Space Flight Center.

Nelson, R. (2007) IT project management: Infamous failures, classic mistakes, and best practices, *MIS Quarterly Executive,* 6, 2, 67-78.

Norton, M. I., Mochon, D. and Ariely, D. (2012) The IKEA effect: When labor leads to love, *Journal of Consumer Psychology,* 22, 3, 453-460.

Nouri, P., Jamali, B. and Ghasemi, E. (2012) The role of experience on techno-entrepreneurs' decision making biases, *Management Science Letters,* 2, 6, 1957-1964.

Pezzo, M. V., Litman, J. A. and Pezzo, S. P. (2006) On the distinction between yuppies and hippies: Individual differences in prediction biases for planning future tasks, *Personality and Individual Differences,* 41, 7, 1359-1371.

Ronen, B. and Pass, S. (2008) Focused operations management: Achieving more with existing resources. Hoboken, NJ: John Wiley & Sons.

Rust, R. T., Thompson, D. V. and Hamilton, R. W. (2006) Defeating feature fatigue, *Harvard Business Review,* 84, 2, 98-107.

Schmidt, R., Lyytinen, K., Keil, M. and Cule, P. (2001) Identifying software project risks: An international delphi study, *Journal of Management Information Systems,* 17, 4, 5-36.

Standish Group. (2009) CHAOS summary 2009, Boston, MA: The Standish Group.

Thaler, R. (1980) Toward a positive theory of consumer choice, *Journal of Economic Behavior & Organization,* 1, 1, 39-60.

Umapathy , K., Purao , S. and Barton , R. R. (2008) Designing enterprise integration solutions: Effectively, *European Journal of Information Systems,* 17, 5, 518-527.

Westfall, L. (2005) The what, why, who, when and how of software requirements, *Proceedings of the ASQ World Conference on Quality and Improvement,* May 16-18, Seattle, WA, 97-105.

Wetherbe, J. C. (1991) Executive information requirements: Getting it right, *MIS Quarterly,* 15, 1, 51-65.