Analyzing the Decision Criteria of Software Developers Based on Prospect Theory

Kanako Kina, Masateru Tsunoda
Department of Informatics
Kindai University
Higashiosaka, Japan
tsunoda@info.kindai.ac.jp

Hideaki Hata
Graduate School of Information Science
Nara Institute of Science and Technology
Ikoma, Japan
hata@is.naist.jp

Haruaki Tamada
Faculty of Computer Science and Engineering
Kyoto Sangyo University
Kyoto, Japan
tamada@cse.kyoto-su.ac.jp

Hiroshi Igaki
Faculty of Information Science and Technology
Osaka Institute of Technology
Hirakata, Japan
hiroshi.igaki@oit.ac.jp

***Abstract*—To enhance the quality of software, many software development support tools and software development methodologies have been proposed. However, not all proposed tools and methodologies are widely used in software development. We assume that the evaluation of tools and methodologies by developers is different from the evaluation by researchers, and that this is one of the reasons why the tools and methodologies are not widely used. We analyzed the decision criteria of software developers as applied to the tools and methodologies, to clarify whether the difference exists or not. In behavioral economics, there are theories which assume people have biases, and they do not always act reasonably. In the experiment, we made a questionnaire based on behavioral economics, and collected answers from open source software developers. The results suggest that developers do not always act to maximize expected profit because of the certainty effect and ambiguity aversion. Therefore, we should reconsider the evaluation criteria of tools such as the f-measure or AUC, which mainly focus on the expected profit.**

Keywords—evaluation criteria; fault prediction; behavioral economics; human factor

I. INTRODUCTION

Recently, software development projects have become larger and more complex, and the deadlines for such projects are often tight. In this situation, many software development support tools and software development methodologies have been proposed to suppresss the failure of a software project (e.g., low quality of software and delay of delivery). For example, many prediction models of fault-prone modules have been proposed [12]. These models predict software modules which have faults, and support makingtesting plans. Also, static code analysis tools have been proposed to support finding faults [5]. These tools analyze software codes, and warn developers of spots which may violate coding standards.

However, not all proposed tools and methodologies are widely used in software development. This may be because applying the tools and methodologies sometimes requires increased costs, and it is not always easy to apply such tools and methodologies to development. Also, they are not always useful for some developers, or developers may not be familiar with them. For example, static code analysis tools are not widely used [2], and even UML is not used widely [13]. We assume that the developers' evaluation of the tools and methodologies is different from the evaluation by researchers.

As an example, we use the performance evaluation of fault-prone module prediction. The prediction of the model includes false positives and false negatives. False positives mean the model indicates that a module includes faults, but it does not actually. False negatives mean the model indicates that a module does not include faults, but it actually does. Researchers evaluate the performance of the prediction model by the f-measure or AUC (area under the curve) [7], which are calculated based on the false positives and the false negatives. However, if developers only focus on the false positives, the evaluation by developers is different from researchers. For instance, a prediction model A is better than a model B based on the f-measure, but the false positives of model B are better than in model A. In this case, developers will highly evaluate model B. However, focusing on the f-measure is more reasonable than focusing on the false positives when the cost due to the false positives is the same as (or smaller than) the false negatives. Therefore, the sum of the false positives and the false negatives (i.e., the sum of the cost) is smaller when the f-measure is better.

Some papers have suggested that the false positives affect the usability of the static code analysis tools. Johnson et al. [10] investigated the reasons why static code analysis tools are not widely used. For example, tool output, supporting teamwork, user input, and customizability are the reasons. Tool output represents a false positive (i.e., although the tool suggests a spot is problematic, actually it is not). Another study [3] also focused on false positives in static code analysis tools. Similarly, developers often focus on the false positives or the

false negatives, and not on the f-measure, when evaluating the fault-prone module prediction.

In this paper, we analyzed the decision criteria of software developers as applied to the tools and methodologies, to clarify whether a biased evaluation is observed or not. In behavioral economics, there are theories which assume people have biases, and do not always act reasonably. Our analysis is based on prospect theory [11]. Prospect theory is used in behavioral economics, and explains how people make decisions. In the software engineering domain, there are a few studies which have applied behavioral economics [8]. However, these studies did not use prospect theory in the analysis.

In this analysis, we conducted a survey e-mailed to open source software developers, and tried to clarify whether developers have the biases illustrated in prospect theory . The result will be useful in reconsidering the evaluation criteria of development support tools and methodologies.

II. PROSPECT THEORY

Our analysis is based on the prospect theory [11], which is one of the theories in behavioral economics. Behavioral economics clarifies the behavior of people, based on experiments and observations. Traditional economics assumes people act rationally. In contrast, behavioral economics assumes people have biased thinking, and they do not always act rationally. Prospect theory explains how people make decisions when there are options which bring profits with different probabilities. The assumptions of prospect theory are different from expected utility theory [14] in traditional economics.

Allais paradox: Prospect theory is proposed to solve the paradoxes of expected utility theory. We explain two paradoxes which we analyzed in our study. The first one is the Allais paradox [1]. Simply speaking, the paradox can be explained by the following example.

Q1. Which lottery would you choose?

A1-1. You get $1,000 with a 100% possibility.

A1-2. You get $2,000 with a 50% possibility, and get $0 with a 50% possibility.

Many people select lottery A1-1 although the expected profit (i.e., the multiplication of the profit and the possibility) of A1-1 is the same as A1-2. People choose lottery A1-1 because it brings profit certainly. People tend to avoid the risk of losing the profit by selecting A1-2. That is, people make a decision, considering not only the expected profit, but also the certainty. This observation is called the **certainty effect** [15].

We focus on the certainty effect, and analyze whether it is observed or not in the developers' decision. If the certainty effect is observed, the evaluation criteria of the development support tools and methodologies may have to be reconsidered. For example, in the two development support tools shown in the following, developers select a tool from between them.

A2-1. A tool which always reduces 10 hours ofworking time.

A2-2. A tool which reduces 25 hours of working time with a 50% possibility, and does not reduce working with a 50% possibility.

Considering the expected profit of the tools, tool A2-2 is better than tool A2-1. As explained in the Introduction, researchers often evaluate the effects of the tools and methodologies, especially the prediction models, based on the expected profits such as in the f-measure or AUC. Based on their criteria, however, researchers would select tool A1-1 whilemany developers, on the other hand, may select tool A2-1, if there is a certainty effect.

Ellsberg paradox: The other paradox of expected utility theory is the Ellsberg paradox [4]. Intuitively, the paradox can be explained by the following example.

Q3. Which jar would you choose, if you get $1,000 when you draw a red ball?

A3-1. An jar which includes 50 red balls and 50 black balls.

A3-2. An jar which includes red balls and black balls. The total number of balls is 100, and the number of red balls and black balls are unknown.

In the example, the profit is not certain, despite choosing A3-1 or A3-2. However, the probability of A3-1 is known, and that of A3-2 is unknown. A3-2 is called an ambiguity. Many people choose A3-1 to avoid the ambiguity. This observation is called as the **ambiguity aversion**.

We focus on the aversion, and analyze whether it is observed or not in the developers' decision. If the ambiguity aversion is observed, the promotion of the tools and the methodologies may have to be reconsidered by thedevelopers. For example, there are two development support methodologies shown in the following, and developers must select a methodology from between them.

A4-1. A methodology which reduces 2 hours of working time with a 80% possibility, and reduces 1 hour with a 20% possibility.

A4-2. A methodology which reduces 4 hours of working at best. The probability is unknown.

Suppose that A4-1 is the methodology already applied by developers. In contrast, A4-2 is a new methodology evaluated by a certain development project. However, most developers have not applied the methodology to their project before. The effect of the new methodology is often unknown, and hence, it is regarded as containing the ambiguity. In this case, the maximum effect of A4-2 is larger than A4-1. However, many developers may select methodology A4-1, due to ambiguity aversion.

TABLE I. DETAIL OF THE QUESTIONNAIRE

Question	Related to	Item
1	Certainty effect	a) A tool which always reduces 2 hours of working time. b) A tool which reduces 5 hours of working time at 50% probability, and does not affect working time at 50% probability.
2		a) A tool which always reduces 2 hours of working time. b) A tool which reduces 4 hours of working time at 70% probability, and increases 1 hour at 30% probability.
3		a) A tool which always reduces 2 hours of working time. b) A tool which reduces 6 hours of working time at 90% probability, and increases 4 hours at 10% probability.
4	Ambiguity aversion	a) A tool which reduces 4 hours of working time at 70% probability, and increases 1 hour at 30% probability. b) A tool which reduces 2 hours of working time at best, and does not affect working time at the worst. The probability is unknown.
5		a) A tool which always reduces 2 hours of working time. b) A tool which reduces 6 hours of working time at best, and increases 1 hour at the worst. The probability is unknown.
6		a) A tool which always reduces 2 hours of working time. b) A tool which reduces 5 hours of working time at best, and does not affect working time at the worst. The probability is unknown.

III. EXPERIMENT

A. Research Questions

In the experiment, we made a questionnaire and collected answers from open source software developers, to clarify three research questions.

- **RQ1**: Are there certainty effects among software developers when selecting software development support tools?
- **RQ2**: Are there ambiguity aversions among software developers when selecting software development support tools?
- **RQ3**: Is the answer chosen by a developer different from the answer chosen as a manager?

If the answer of RQ1 is "yes," the evaluation criteria of the tools and methodologies should be reconsidered by researchers, especially criteria such as the f-measure and AUC. Also, if the answer of RQ2 is "yes," then how to promote the tools and the methodologies to developers must be reconsidered. For example, if there is some ambiguity in applying the tools, then the effects of the tools are large enough to ignore the ambiguity.

In questions relating to RQ1 and RQ2, we assume that the profit of the tools for developers is the reduction of effort (time). In questions relating to RQ3, the subjects assume that their role is as a project manager, and the profit is an evaluation of themselves from other project's members. This is because the tools and the methodologies are evaluated not only by developers, but also by project managers.

B. Experimental Setup

We e-mailed the survey to open source software developers. Their addresses were collected from websites [6][9]. The answers to the questionnaire were collected via a web form. The number of responses was twenty, and the average number of years of their experience in software development was 21.2 (The median was 20.0). Therefore, the subjects were considered to understand software development very well.

The questionnaire consists of six questions. Details of the questionnaire are shown in Table I. Subjects selected coding support tool 'a' or 'b,' assuming the following.

- 10 hours are needed to make a module.
- Making a module is not hobby programming.
- Conditions such as the time needed to learn a tool are the same for each tool.
- The tools are prediction tools such as a static code analysis tool.

The last item was selected to emphasize that the questionnaire does not focus on the functions of the tools, but on the prediction performance of the tools.

The questions 1 to 3 relate to the certainty effect, i.e., RQ1. In the questions, tool 'a' certainly reduces working time. Although the effect of tool 'b' is not certain, the expected profit (i.e., reduced time multiplied by the probability) is higher than tool 'a.' To analyze the decision criteria of developers in detail, we specified the characteristics of tool 'b' as follows:

- Question one: There is no risk if the tool does not work well.
- Question two: There is a small risk if the tool does not work well.
- Question three: There is a high risk if the tool does not work well although the probability is low.

Questions 4-6 relate to ambiguity aversion, i.e., RQ2. In the questions, the probability of the effect of tool 'a' is known, but that of the tool 'b' is unknown. We specified the characteristics of tool 'a' and 'b' as follows:

- Question 4:

Tool ‘a’ - There is a small risk if the tool does not work well.
Tool ‘b’ - There is no risk if the tool does not work well.

- Question 5:
  Tool ‘a’ - There is no risk if the tool does not work well.
  Tool ‘b’ - Developers receive a high benefit if the tool works well, but there is small risk if it does not work well.
- Question 6:
  Tool ‘a’ - There is no risk if the tool does not work well.
  Tool ‘b’ - Developers receive a high benefit if the tool works well, and there is no risk if it does not work well.

Also, subjects selected coding support tool ‘a’ or ‘b’ from questions 1-6, assuming the following. The questions relate to RQ3.

- You are a manager, and give a tool to a programmer.
- If his/her work time is more than 10 hours, the evaluation of your team is down.

Therefore, the number of answers by each subject is 12. That is, we asked the same set of questions twice. One set is for the role of the developer, and the other one is for the role of the manager. The former set is denoted by Q1-1 to Q1-6, and the latter set is denoted by Q2-1 to Q2-6.

C. Results

The ratio of answers (a) and (b) is shown in Table II. In the following, we explain the analysis results on each research question.

The result related to RQ1: On Q1-1 to Q1-3, the ratio of answer (a) was higher than answer (b). As explained before, the expected profit (i.e., the reduced time multiplied by the probability) of answer (b) is higher than that of answer (a). Therefore, the result means developers do not always act to maximize the expected payoff, and the certainty effect exists when developers select software development support tools. That is, the answer to the RQ1 is “yes.”

In Q1-1, many developers did not select the tool with a relatively high expected profit and no risk, if it did not work well. Also, in Q1-3, the developers did not select the tool with a high expected profit where the probability of the risk was low. Consequently, we must reconsider the evaluation of the tools, if the tools have such characteristics.

The result related to RQ2: On Q1-4 and Q1-5, the ratio of answer (a) was higher than answer (b), but on Q1-6, the ratio of answer (a) was lower than answer (b). On Q1-4 especially, tool ‘a’ was selected by many developers, although tool ‘a’ has risk and tool ‘b’ has no risk. Thus, the result suggests that the ambiguity aversion exists when developers select software development support tools.

TABLE II. THE RATE OF EACH ANSWER ON THE QUESTIONNAIRE

Questions	Answer (a)	Answer (b)
Q1-1	65%	35%
Q1-2	70%	30%
Q1-3	65%	35%
Q1-4	55%	45%
Q1-5	65%	35%
Q1-6	45%	55%
Q2-1	60%	40%
Q2-2	70%	30%
Q2-3	70%	30%
Q2-4	50%	50%
Q2-5	60%	40%
Q2-6	45%	55%

In contrast, tool ‘b’ was selected although the probability of the effect is unknown on Q1-6. This means that when the tool is very effective and has no risk, it will be accepted by developers, even if the probability of the effect is unknown. Note that although the effect of tool ‘b’ on Q1-5 is similar to the effect on tool ‘b’ on Q1-6, many developers did not select tool ‘b’ on Q1-5. This is because tool ‘b’ on Q1-5 has a small risk and hence, developers avoided selecting it. Consequently, the answer to RQ2 is “yes, except for when the tool is very effective and has no risk.”

The result related to RQ3: Overall, the ratio of answers (a) and (b) on Q2-1 to Q2-6 was similar to Q1-1 to Q1-6. Although the ratio of answers (a) and (b) was same on Q2-4, the ratio of answer (a) was higher than that of answer (b) on Q2-5 (tool ‘b’ on Q2-4 has no risk, and tool ‘b’ on Q2-5 has a small risk). Hence, the ambiguity aversion still exists when a tool has a small risk. Based on this result, we concluded that the answer to RQ3 is “No.” That is, the decision criteria of project managers is almost the same as the developers.

IV. CONCLUSION

This paper analyzed the decision criteria of software developers to software development support tools and methodologies in order to reconsider the evaluation criteria. In the survey, we analyzed whether the certainty effect and the ambiguity aversion exist or not on the developers’ decisions. Based on the analysis results, we suggest the following:

- Developers do not always act to maximize expected profit because of the certainty effect. Therefore, we should reconsider the evaluation criteria of tools such as the f-measure or AUC, which mainly focus on the expected profit.
- Developers avoid selecting tools if the probability of the effect of the tools is unknown, and the tools have some risks. This result suggests that it is not easy to apply new tools to actual software development projects since the probability of the effect is generally

unknown before the application. To promote development support tools, we have to suppress the risk of the tools.

- The decision tendency of the project managers was almost the same as the developers. Therefore, the role of the person making the decision may not be very important when promoting the tools and methodologies.

We believe that the main contribution of our paper is its quantitative clarification of the decisions of the software developers described above.

ACKNOWLEDGMENTS

This research was partially supported by the Japan Ministry of Education, Science, Sports, and Culture [Grant-in-Aid for challenging Exploratory Research (No. 26540029), Grant-in-Aid for Scientific Research (C) (No. 25330090), and Grant-in-Aid for Young Scientists (B) (No. 24700030)].

REFERENCES

[1] M. Allais, “Rational man's behavior in the presence of risk: critique of the postulates and axioms of the American school,” *Econometrica*, vol.21, pp.503-546, 1953.

[2] N. Ayewah, D. Hovemeyer, J. Morgenthaler, J. Penix, and W. Pugh, "Using Static Analysis to Find Bugs," *IEEE Software*, vol.25, no.5, pp.22-29, 2008.

[3] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler, “A few billion lines of code later: using static analysis to find bugs in the real world,” *Communications of the ACM*, vol.53, no.2, pp.66-75, 2010.

[4] D. Ellsberg, “Risk, Ambiguity, and the Savage Axioms,” *Quarterly Journal of Economics*, vol.75, no.4, pp.643-669, 1961.

[5] FindBugs, http://findbugs.sourceforge.net.

[6] The "Fossies" Software Archive, http://fossies.org/.

[7] T. Hall, S. Beecham, D Bowes, D. Gray, and S. Counsell, “A Systematic Literature Review on Fault Prediction Performance in Software Engineering,” *IEEE Transactions on Software Engineering*, vol.38, no.6, pp.1276-1304, 2012.

[8] R. Hofman, “Behavioral economics in software quality engineering,” *Empirical Software Engineering*, vol.16, no.2, pp.278-293, 2011.

[9] J. Howison, M. Conklin, and K. Crowston, “FLOSSmole: A collaborative repository for FLOSS research data and analyses,” *International Journal of Information Technology and Web Engineering*, vol.1, no.3, pp.17–26, 2006.

[10] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, “Why don't software developers use static analysis tools to find bugs?” *In Proc. of the International Conference on Software Engineering (ICSE)*, pp.672-681, 2013.

[11] D. Kahneman and A. Tversky, “Prospect Theory: An Analysis of Decision under Risk,” *Econometrica*, vol.47, no.2, pp.263-91,1979.

[12] O. Mizuno and T. Kikuno, “Training on errors experiment to detect fault-prone software modules by spam filter,” *In Proc. of the joint meeting of the European software engineering conference and the symposium on The foundations of software engineering (ESEC-FSE)*, pp.405-414, 2007.

[13] M. Petre, “UML in practice,” *In Proc. of the International Conference on Software Engineering (ICSE)*, pp.722-731, 2013.

[14] P. Schoemaker, Experiments on Decisions under Risk: The Expected Utility Hypothesis, Springer Netherlands, 1980.

[15] A. Tversky and D. Kahneman, “Rational Choice and the Framing of Decisions,” *Journal of Business*, vol.59, no.4, pp.S251-S278, 1986.