The Importance of NOT Learning from Experience

Magne Jørgensen
Industrial Systems Development, Department of Informatics, University of Oslo, Norway

Dag Sjøberg
Industrial Systems Development, Department of Informatics, University of Oslo, Norway

1 Introduction

> *"What we learn from history is that people don't learn from history."*
> (George Bernard Shaw)

In this paper we argue that learning correctly from software development experience can be a difficult task and that, frequently, there is not enough information available to draw valid conclusions. In our opinion, a focus on how to learn from experience should include a focus on how to avoid incorrect learning from experience. This includes a focus on when we should **not** learn from experience.

We have tried to transfer general findings from human judgement studies and other sources to a software context and, in particular, to an experience database context. We believe that it is especially important to avoid incorrect reasoning from experience when establishing experience databases to spread experience within an organisation. The lack of focus on how to avoid incorrect reasoning may be a major reason for, as far as we have observed, the lack of success of implementation of software development experience databases. If we cannot trust the contents of an experience database, or the contents do not lead to better judgement processes, then experience databases are not very useful.

The remainder of this paper is organised as follows. Section 2 gives an example illustrating problems we face when attempting to learn from experience. Section 3 discusses typical learning from experience fallacies and techniques to avoid them. Section 4 concludes and describes further work.

2 An example

A project leader summarises lessons learned from previous projects that he has lead. The project leader remembers that most (approx. 80%) of his projects with time overruns had an incomplete requirement specification (at the time when the design phase was started). The project leader, consequently, draws the conclusion that an incomplete requirement specification means a high risk of late delivery.

This conclusion may look rather obvious, but is in many ways a typical example of high risk of incorrect learning from experience:

1) If the incomplete requirement specification is not the cause, only a correlating factor, then the observed relation may not be present in the future. In this example, a more likely reason for late delivery is not the incomplete requirement specification alone, but the combination of incomplete requirement specification, use of project models assuming fixed and complete requirements (for example, a waterfall model) and lack of proper requirement management. Looking at only one of these variables in isolation may not be meaningful. This error is the frequently made "preceding is causing"-error, see [4].
2) If the project leader strongly believes that incomplete requirement specification is a major reason for project delays, this will impact how he interprets project events. For example, the classification of requirement specifications into incomplete and complete can be a highly subjective task. In particular, this classification can be biased when it is carried out after the completion of the projects. This error is an example of "selective perception", discussed in Section 3.
3) It is not sufficient to remember the projects with incomplete requirement specifications and their proportion of "time overruns". It is just as important to remember the projects **without** incomplete requirement specifications and their proportion of "time overruns"! Assume that the project leader has carried out 18 projects distributed as follows:

	Incomplete specification	Complete specification
Time overrun	10 projects	2 projects
On time	5 projects	1 project

Table 1 – Project distribution

The project leader correctly summarises that approx. 80% (10/(10+2)) of the projects with time overruns had an incomplete requirement specification. This does, however, not mean that an incomplete specification means a higher risk of time overruns. The frequency of time overruns given a complete requirement specification (2/(2+1) = 67%) is exactly the same as the frequency of time overruns given an **incomplete** requirement specification (10/(10+5) = 67%). In other words, the completeness of the requirement specification is of no use to predict time overrun. This error is an example of the "neglect of base rates"-error [4]. We do not remember the situations where an incomplete requirement specification is not followed by a time overrun as well as the situations where an incomplete requirement was followed by time overruns, and/or do not use the available information about the frequency of the scenarios properly.

The project leader has, of course, much more information regarding the consequences of the incomplete requirement specifications in each projects, for example, the degree

of rework needed and the frustration of the project members. This detailed scenario information can be useful, but can also lead to "representative thinking" [19] and other "learning from experience"-problems discussed in Section 3. Contrary to what our "intuition" tells us, more information does frequently lead to worse explanation or prediction models [19].

3 How to avoid incorrect learning from experience

This section discusses a small, but important, selection of "learning from experience"-problems and how to avoid them. The selection and discussion are far from complete and we recommend [1, 2, 4, 7, 8, 10, 15, 16, 19] to get a more complete picture.

3.1 Creation of experience

Selective perception: The same events are perceived differently by different people. Studies show that what we perceive is heavily influenced by what we expect and want to find [19, 24]. For example, a study [11] of how different football supporters perceived a football game concluded: *"It seems clear that the 'game' actually was many different games. ... It is inaccurate and misleading to say that different people have different 'attitudes' concerning the same 'thing'. For the 'thing' simply is not the same for different people whether the 'thing' is a football game, a presidential candidate, communism, or spinach*". Transferred into a context of software development experience database, this means, for example, that a software project experience database should reflect differences in perceptions. Use of multiple perspectives and an acceptance that a software project is not (opinions on) the same thing for all actors may reduce the risks of selective perception. The experience collector should ask the experience providers to consider reasons why their perception may be selective or biased.

Hindsight bias: When people know the actual outcome of a process, they tend to regard that outcome as having been fairly predictable all along – or at least more predictable than they would have judged before knowing the outcome [17]. Experience from software projects collected after their completion can be strongly impacted by hindsight bias ("I-knew-it-from-the-beginning..."). Unfortunately, it is not enough to inform people about hindsight bias and encouraging them to avoid it [9]. To reduce the size of the hindsight bias, people should be asked to consider how alternative outcomes might have occurred.

Improper learning models:

Several studies [1, 6, 14] report a disappointingly low correlation between length of experience and the quality of professional's judgements. An improper learning model is one important reason for this [1]:

- We try to confirm theories, rather than reject incorrect hypotheses.
- The fact that we are able to formulate a rule is often sufficient to believe that we have a good rule even though we have no experience indicating that the rule is valid. That is, the confidence in own knowledge increases with the ability to find rules regardless of the validation of these rules.

- In cases where we act on the experience based judgement there will be a number of additional factors that prevent us from detecting that our judgement is incorrect, e.g. self-fulfilling prophesies.
- We tend to prefer deterministic rules (IF <a> THEN <b> - type of rules) even if the relationship between variables is probabilistic (IF <a> THEN 20% probability of <b> - type of rules). If we find no deterministic rules, we tend to assume that there is no rule at all and start guessing. Applied on software projects this means that we tend to prefer rules where the same software project characteristics lead to the same project outcomes each time, see [15]. We do this in spite of the fact that a probabilistic rule where similar software projects may have several possible outcomes, each of them with a connected probability, would be more appropriate. The very limited use of probabilistic software effort estimation models in the studied organisation supports that this preference of deterministic rules is the case for maintenance work, too.

A key to improve learning from experience in a probabilistic environment is, according to [1] and [10], that we are able to detect the probabilistic nature of probabilistic tasks. However, the only way a person can detect this is to evaluate the usefulness of deterministic rules against probabilistic rules. This requires that the person must have the hypothesis that the task may be probabilistic and know how to test for this. This knowledge can, according to [1], hardly be derived from experience alone, but must be taught. Proper education and proper concepts of probability based relations may thus be necessary in order to learn from experience in a highly probabilistic environment [10].

Usefulness of experience: Much of our experience is of no value for future work:

- Change of conditions makes history invalid. Sometimes the experience itself changes the conditions so much that the experience is invalid. The change of conditions is particularly frequent in a software development context.
- Frequently, it is impossible to compare what actual happened with what had happened if the actual actions were not taken. A cause effect is, therefore, very difficult in one of a kind development projects.
- Experience is context dependent and much of the context is hard (impossible?) to describe. One of the authors of this paper once tried to extract useful information from a number of project experience reports in a large Norwegian company, but found that the lack of context information made the use of those reports for other projects impossible [25].

It is important to reflect on the usefulness of experience and try not to be lead by our intuition of how important our experience is. Our intuition of the usefulness of our own experience is easily impacted by irrelevant factors, such as, the degree of effort, amount of frustrations or the emotions involved. We should accept that sometimes there is nothing or very little to learn from experience, even when we have worked hard and a lot of very interesting events have occurred.

3.2 Use of experience

Software development environments are complex, and, frequently, we have not the time, the competence and/or possibility to collect and process all information necessary to make 100% valid conclusions. We therefore use “heuristics” (rules of thumb) based on the available experience and information to make decisions or

extract general rules at a, hopefully, sufficient level of quality. Most of the heuristics are unconscious and, for this reason, hard to analyse and change by the subjects themselves.

Availability heuristic: One frequently used heuristic is the "availability heuristic" [22], which states that we *"assess the frequency of a class or the probability of an event by the ease with which instances or occurrences can be brought to mind"*. Usually, this heuristic works quite well, common events are easier to remember than uncommon events. There are, however, situations where this heuristic leads to incorrect judgements. Some events are easier to remember because they are inherently easier to think of, because they have taken place recently or because they are highly emotional. The improper use of this heuristic may be the main reason for our finding in [13], where the software managers believed that their maintenance groups use much more effort on software corrections than the groups actually did. Software corrections, probably, receive more management attention compared with other types of maintenance work and are, therefore, overestimated.

One way to avoid the problems connected with the availability heuristic may be to keep records of the software development work and frequently update the records. Post-mortem project reviews (experience reports) based on what the project members remember at the end of the projects may be better than no experience collection. On the other hand, if no experience is recorded during the projects (for example, in a project diary) much of the information collected may be very much biased due to the availability heuristic.

Anchoring and adjustment: Anchoring and adjustment may be the most common heuristic when reasoning by analogy. This heuristic consists of two steps:

1) Make an initial judgement (an "anchor").
2) Adjust the initial judgement for differences between current situation and the "anchor" situation.

Studies [18, 22] indicate that the adjustments are, most of the time, insufficient and that the anchor has an unexpected high impact on the final judgement. This heuristic is used when experts are estimating software project effort. The estimation experts find the closest analogies (the anchor projects), for example using the "representativeness heuristic" (see explanation later in this section). Then the estimation expert adjusts for differences between the anchor projects and the project to be estimated.

As found in [12], an estimation process based on "anchoring and adjustment" (expert estimation) performs well compared with other estimation methods when the anchor projects are good predictors of the new project. This process leads, however, to lower accuracy than more statistically based estimation models when the anchor projects is quite different from the project to be estimated. The reason for this low accuracy is the inability of expert estimators to adjust sufficiently for large differences in project characteristics.

The problems people have in adjusting insufficiently from anchor values seem to be a very robust effect that is difficult to prevent. Frequently we are unaware of what we use as anchor values. The maybe only way to avoid the problems and, at the same time, keep the advantages of this heuristic is to increase awareness of the heuristic and to know when to use more formal approaches, such as statistically based prediction methods.

Interestingly, the knowledge of this human weakness (that we do not adjust sufficiently from the anchor based value) can be used to compensate for another

human weakness; the tendency for planners to make too optimistic prediction. The compensation technique, used by some project leaders, is to start the estimation process with the "most pessimistic" effort estimate and then estimate "most likely" and "most optimistic" effort. The "most pessimistic" effort estimates will then be the "anchor values" and lead to lower estimates of "most likely" effort compared with an estimation process starting with the "most likely" effort estimates. Results supporting this finding are reported in [3].

Representativeness heuristic: Another common heuristic is the "representativeness" heuristic [22], i.e., a judgement process based on the similarity between A and B and the assumption that similarity with respect to some properties means similarity with respect to other properties. This heuristic is a type of analogy based reasoning. The representativeness heuristic works well in most cases, but can be a major source of biased use of experience in some cases, as reported in [23], p 98:

"*As the amount of detail in a scenario increases, its probability can only decrease steadily, but its representativeness and hence its apparent likelihood may increase. The reliance on representativeness, we believe, is a primary reason for the unwarranted appeal of detailed scenarios and the illusory sense of insight that such constructions often provide.*"

The results we describe in [14] exemplify this. Interviews with software maintainers indicated that they used the representativeness heuristic together with the "anchor and adjust" heuristic when predicting problems connected with solving a software maintenance task. The maintainers tried to remember a similar task carried out recently (the anchor) and based their predictions on the degree of problems conducting that task adjusting for differences. This heuristic did not work well for the software maintainers predicting maintenance problems. Although they had detailed knowledge about the task to be completed by themselves, a simple decision model based solely on the historical rate of major problems for large/medium and small maintenance tasks outperformed the prediction of the maintainers. Similar results are reported in, for example, [5].

The representativeness heuristic is based on (or influenced by) the assumption: "The more similar A is to B, the more likely A will behave similarly to B". This assumption may be correct most of the time and is a very useful heuristic. To avoid the misuse of this heuristic we must, however, understand the limitations. For example, consider a new software project A, which is very similar to a completed project B, which in turn had an extremely high productivity. This extremely high productivity is not only caused by the variables that make project A similar to project B, but also to other variables, such as an unusual degree of luck. Then, it is unlikely that project A will have the same high productivity as project B (regression to the mean-effect [21]). In this case, it would have been better to use less similar projects as a basis for the estimation of the productivity of project A.

3.3 General guidelines

The suggestions on how to achieve the goal of not learning from experience can be summarised as:

- Ask the experience providers to consider alternative perspectives. Stimulate critique of own judgements. (This guideline is, in our opinion, the most

important and robust guideline on how to avoid biases and misuse of the heuristics.)

- Learn about the biases and the limitations of the heuristics and increase the awareness of them. Note, however, that learning about the biases and limitations of the heuristics is not always sufficient to eliminate them.
- Conduct training (and give feedback) on how to avoid the biases and the misuse of the judgement heuristics.
- Use statistical techniques and increase the statistical knowledge of the software professionals. In particular, skills on how to establish probability-based models of relationship between project characteristics are important, we believe.

4 Conclusions and further work

Learning correctly from experience is difficult. Without knowledge and reflections on the biases and limitations of the judgement heuristics:

- much of the learning will be incorrect,
- experience databases and "post mortem reviews" (experience reports) will include much incorrect information, and
- invalid conclusions about causal relationships will be drawn.

The perhaps simplest and most efficient way to reduce the negative impact of the biases and judgement heuristics is to ask the software professionals to consider alternative perspectives and to stimulate critique of own judgements. In addition, we need to increase the awareness and knowledge of software professionals and researchers.

The problems of learning from experience have been studied for at least 60 years. Surprisingly, the work on experience databases in software organisations does not refer to or incorporate the knowledge from these studies. We believe that it should.

This paper described the beginning of our effort on how to improve learning from experience through tools and processes based on knowledge about biases and judgement heuristics. We have already started collecting data in collaboration with a major Telecom company.

Acknowledgements

We gratefully acknowledge the collaboration with our industrial partner, Ericsson. The research project is funded by The Research Council of Norway through the industry-project PROFIT (PROcess improvement For the IT industry).

References

[1] B. Brehmer, "In one word: Not from experience," *Acta Psychologica*, Vol. 45, 1980.

[2] C. F. Camerer and E. J. Johnson, "The process-performance paradox in expert judgment," in *Toward a general theory of expertise*, K. A. Ericsson

and J. Smith, Eds. Cambridge: Cambridge University Press, 1991, 195-217.
[3] T. Conolly and D. Dean, "Decomposed versus holistic estimates of effort required for software writing tasks," *Management Science*, Vol. 43, 1997, pp. 1029-1045.
[4] R. M. Dawes, *Rational choice in an uncertain world*: Harcourt Brace & Company, 1988.
[5] R. M. Dawes and B. Corrigan, "Linear models in decision making," *Psychological Bulletin*, Vol. 81, 1974, pp. 95-106.
[6] S. M. Doane, J. W. Pellegrino, and R. L. Klatzky, "Expertise in a computer operating system: conceptualization and performance," *Human Computer Interaction*, Vol. 5, 1990, pp. 267-304.
[7] H. J. Einhorn, "Confidence in judgement: Persistence of the illusion of validity," *Psychol. rev.*, Vol. 85, 1978, pp. 395-416.
[8] I. Fischer and N. Harvey, "Combining forecasts: What information do judges need to outperform simple average?," *International journal of forecasting*, Vol. 15, 1999, pp. 227-246.
[9] B. Fischhoff, "Perceived informativeness of facts," *Journal of Experimental Psychology: Human Perception and Performance*, Vol. 3, 1977, pp. 349-358.
[10] K. R. Hammond, *Human judgement and social policy*. New York: Oxford University Press, 1996.
[11] A. H. Hastorf and H. Cantril, "They saw a game: A case study," *Journal of Abnormal and Social Psychology*, Vol. 49, 1954, pp. 129-134.
[12] S. J. Hoch and D. A. Schkade, "A psychological approach to decision support systems," *Management Science*, Vol. 42, 1996, pp. 51-64.
[13] M. Jørgensen, "The quality of questionnaire based software maintenance studies," *ACSM SIGSOFT - Software Engineering Notes*, Vol. 20, 1995, pp. 71-73.
[14] M. Jørgensen and D. I. K. Sjøberg, "Learning from experience in a software maintenance environment," *Submitted to IEEE Transactions on Software Engineering*, 2000.
[15] D. Kahneman and A. Tversky, "On the psychology of prediction," *Psychological Review*, Vol. 80, 1973, pp. 237-251.
[16] D. Kahnemann, P. Slovic, and A. Tversky, *Judgement under uncertainty: Heuristics and biases*: Cambridge University Press, 1982.
[17] M. R. Leary, "Hindsight distortion and the 1980 presidential election.," *Personality and Social Psychology*, Vol. 8, 1982, pp. 257-263.
[18] G. B. Northcraft and M. A. Neale, "Experts, amateurs, and real estate: An ancoring-and-adjustement perspective on property pricing decisions.," *Organizational Behavior and Human Decision Processes*, Vol. 39, 1987, pp. 84-97.
[19] S. Plous, *The psychology of judgment and decision making*: McGraw-Hill, 1993.
[20] P. M. Senge, *The fifth discipline: The art and practice of the learning organization*: Currency/Doubleday, 1995.
[21] W. Trochim, *The Research Methods Knowledge Base*, 2nd ed: Cornell Custom Publishing, Cornell University, Ithaca, New York, 1999.
[22] A. Tversky and D. Kahneman, "Judgment under uncertainty: Heuristics and biases," *Science*, Vol. 185, 1974, pp. 1124-1130.
[23] A. Tversky and D. Kahneman, "Judgments of and by representativeness," in *Judgment under uncertainty: Heuristics and biases*, D. Kahneman, P. Slovic, and A. Tversky, Eds. Cambridge, England: Cambridge University Press, 1982.

[24] R. P. Vallone, L. Ross, and M. R. Lepper, “The hostile media phenomenon: Biased perception and perceptions of media bias in coverage of the Beirut massacre.,” *Journal of Personality and Social Psychology*, Vol. 49, 1985, pp. 577-585.

[25] M. Jørgensen, D.I.K. Sjøberg, and R. Conradi, “Reuse of software development experience at Telenor Telecom Software,” European Software Process Improvement Conference (EuroSPI’98), Gothenburg, Sweden, 16 – 18 November 1998, pp. 10.19–10.31.