Software Process Improvement and Human Judgement Heuristics

Magne Jørgensen[i], Dag I. K. Sjøberg[i,ii],
i) Department of Informatics, University of Oslo, Norway,
ii) Simula Research Laboratory, Norway

Abstract

This paper exemplifies how better knowledge about the human judgement strategies known as heuristics can be used to improve software processes. Human judgement heuristics work well when they exploit a fit between their structure and the structure of the environment in which they are used. This use of environmental fit may lead to amazingly good judgements based on little information and simple computations compared with more formal approaches. Sometimes, however, the heuristics may lead to poor judgements. Knowing more about the strengths and weaknesses of human judgement heuristics we may be able to (1) know when to use formal process improvement approaches and when to use less expensive expert judgements, (2) support the experts in situations where the experts' judgements strategies are known to perform poorly, (3) improve the formal processes with elements from the experts' strategies, and (4) train the experts in the use of more optimal judgement strategies. A small-scale experiment was carried out to evaluate the use of the representativeness heuristic in a software development effort estimation context. The results indicate that the actual use of the representativeness heuristic differed very much among the estimators and was not always based on an awareness of fit between the structure of the heuristic and the structure of the environment. Estimation strategies only appropriate in low uncertainty development environments were used in high uncertainty environments. A possible consequence of this finding is that expert estimators should be trained in assessing how well previous software projects predict new software projects, i.e., the uncertainty of the environment, and how this uncertainty should impact the estimation strategy.

Key words: Software process improvement, human judgement heuristics, expert judgement, software effort estimation.

1. Introduction

An important principle of Scientific Management (Taylorism) is to decrease the reliance on expert judgement and to increase the use of well-defined work processes derived from measurement and statistical analyses. In fact, the main purpose of Taylor's book *The Principles of Scientific Management* (Taylor, 1911) is «to show the enormous gains which would result from the substitution by our workmen of scientific for rule-of-thumb methods.» Implementation of this principle has typically had the following consequences:

- A work process improvement group, process owners or the managers get the formal responsibility for improvement of the process, not the workers.
- The work processes are defined and measured.
- The work process variance is controlled and analysed.
- Work process changes are managed as controlled experiment, with scientific analyses of the causes and/or impacts of process changes.

This principle has, in spite of several anti-Tayloristic initiatives (Pruijt, 1997), proved to be persistent, and there are reasons to believe it has had a strong influence on current software improvement frameworks. For example, process owner, process measurement and statistical process control are important elements of the common process improvement frameworks Total Quality Management (Deming, 2000) and the Capability Maturity Model (Paulk *et al.*, 1993). The increased reliance on scientific methods to improve software processes is, of course, only meaningful when leading to better processes than the processes developed or evolved applying informal expert judgement. We probably need both formal and informal software process improvement approaches. In particular, we need to know the cost and benefits of different approaches. To do this we need to know how and how well software professionals judge. If we knew the judgement strategies used by software professionals, we may be able to:

- know when to use formal process improvement approaches and when to use less expensive expert judgements,
- support the experts in situations where the experts' strategies are known to perform poorly,
- improve the formal processes with elements from the experts' strategies, and
- train the software professionals in the use of more optimal judgement strategies.

Unfortunately, as far as we know, neither the Scientific Management literature nor the software process improvement literature describes how to achieve these goals. On the other hand, there have been a number of relevant psychological studies on human judgement. These studies are not referred to in the software process improvement literature. The purpose of this paper is, therefore, to exemplify how we may apply results from psychological studies on how people judge to improve software processes.

Our work is part of a Norwegian software process improvement research project: PROFIT (PROcess improvement For the IT industry). PROFIT is a 3 year project running from 2000 until 2003 and is funded by the Research Council of Norway (NFR). Its main goal is to increase the competitiveness and profitability of Norwegian IT-industry through systematic and continuous process improvement. An important aspect of PROFIT is to provide a means for academia and software industry to meet and share software process improvement experiences and research results. The research partners are University of Oslo (UiO), the Foundation for Scientific and Industrial Research at the Norwegian Institute of Technology (SINTEF) and the Norwegian University of Science and Technology (NTNU). Currently, 10 Norwegian software development organisations participate in the project. Through the participation in PROFIT we carry out software improvement studies in industrial environments to get feedback on the importance of the problems we study and on the validity of our research results. The results described in this paper are based on studies in co-operation with the PROFIT software development organisations and an experiment using computer science students.

Although this paper covers software process improvement in general, its main focus is on human judgement heuristics for software project effort estimation. Software project effort estimation is, in our opinion, a good example of a process that can be improved using both formal and informal approaches when one knows more about the judgement strategies of software professionals. The empirical studies on software project effort estimation indicate that formal estimation models are not systematically more accurate than informal expert judgement (Conte *et al.*, 1986, Finnie & Wittig, 1997, Kemerer, 1987, Myrtveit & Stensrud, 1999). On the other hand, there is evidence indicating that a *combination* of expert and model provides more accurate effort estimates than expert and model estimates alone (Briand *et al.*, 1998, Jørgensen, 1997, Myrtveit & Stensrud, 1999). Similar results are found in other domains, such as business forecasting (Blattberg & Hoch, 1990). A reason for the benefits of combining expert-based and model-based estimates is that they have different strengths and weaknesses. Potential strengths of formal estimation models are, for example, that they have less bias towards over-optimistic effort estimates (Jørgensen, 1997) and that the models' effort estimates are less influenced by organisational and social pressure. An example of social pressure, is when a manager gives a project leader positive feedback on low effort estimates and negative feedback on high effort estimates. Potential strengths of an expert judgement are that experts can identify new variables relevant for the actual project effort estimate and can include «broken-leg» situations, i.e., very rare situations that are not meaningful to include in a formal estimation model. While we know much about the properties of estimation models, we do not have much knowledge about expert estimation strategies. According to Brown & Siegler (1993) psychological research on real-world quantitative estimation *has not culiminating in any theory of estimation, not even in a coherent framework for thinking about the process.* We believe that knowledge about the strategies used by expert estimators will enable better use of both expert estimates and estimation models, and ultimately lead to an improved software estimation process.

The remainder of this paper is organised as follows. Section 2 describes the term judgement heuristic and introduces two research approaches on human judgement heuristics, namely the so-called «Heuristics and Biases» and «Fast and Frugal» approaches. Sections 3 and 4 describe these two approaches, respectively. Section 5 illustrates their relevance for software processes. Section 6 describes and discusses a specific experiment analysing the use of the «representativeness heuristic» by computer science students when estimating software development productivity. Finally, Section 7 concludes and describes further work. Table 1 gives an overview of where the different heuristics are covered in this paper.

Research approach	**Heuristic class: Instance**	**Description**	**Software process relevance**
Heuristics and biases	Representativeness	Section 3.1	Section 5.1
	Availability	Section 3.2	Section 5.2
	Anchoring and adjustment	Section 3.3	Section 5.3
Fast and Frugal	Ignorance: Recognition	Section 4.1	Section 5.4
	One reason: Take The Best	Section 4.2	Section 5.5
	Elimination: QuickEst	Section 4.3.1	Section 5.6
	Elimination: Categorisation	Section 4.3.2	Section 5.7

Table 1: Human judgement heuristics covered in this paper

2. Human Judgement Heuristics

The term *heuristic* is of Greek origin and means «to discover» (Collins, 1991). The term has several different interpretations, e.g.:

- *Useful, even indispensable cognitive processes for solving problems that cannot be handled by logic and probability theory* (Groner, 1983).
- *A rule of thumb for solving a problem without the exhaustive application of an algorithm (Collins, 1991).*

Our use of the term *human judgement heuristics* follows the interpretation in Gigerenzer *et al.* (1999): *cognitive processes that do not necessarily lead to an optimal solution*. The cognitive processes we are interested in are the strategies people use to reach decisions in complex environments without the use of sophisticated and computation intensive algorithms. These strategies have to be simple and sufficiently fast to be used by people in real-life situations, e.g., situations where decisions cannot wait until all relevant information is available and has been analysed. As we will illustrate in the next sections, these strategies may lead to biased judgements. On the other hand, they may perform amazingly well compared with much more sophisticated and computation intensive algorithms.

Human judgement relies on *indicators*. An indicator, frequently described as a *cue* in the human judgement literature, is a variable with relevance for the judgement to be made. When for example estimating effort, the known software development project characteristics useful for effort estimation purposes are indicators of the actual effort of the project. Indicators can be more or less *valid* relative to a judgement, i.e., more or less useful for judgement purposes. Different heuristics use different definitions of validity. The *criterion* is the variable to be estimated, e.g., the effort needed in a software project.

In recent years a large amount of research studies have been devoted to uncover the human judgement heuristics, to analyse the contexts when heuristics are used, and to analyse how people learn and adapt heuristics. Overviews of these studies can be found in (Gigerenzer *et al.*, 1999), (Kahnemann *et al.*, 1982) and (Plous, 1993). The research studies point at several heuristics that people may use or, at least, are simple and fast enough for people to be possible to use without any computational support. It is difficult to categorise and give an exhaustive review of the judgement heuristics described in the research literature. The heuristics are not always well specified, with only general labels on a broad range of methods, they may be overlapping and they may be differently described in different research papers. To simplify the presentation, we have based the work in this paper mainly on two competing research approaches, respectively described in (Kahnemann *et al.*, 1982) and (Gigerenzer *et al.*, 1999).

The first approach builds upon an effort initiated by Tversky & Kahnemann (1974) and is known as the «Heuristics and Biases» approach. It emphasises how the use of heuristics may lead to non-optimal solutions compared with more scientific and statistically sound methods. As a reaction on the strong focus on how heuristics lead to judgement fallacies, the Adaptive Behaviour and Cognition Research Group (ABC Group)[1] lead by Gigerenzer was initiated (Gigerenzer & Goldstein, 1996). Their research has a more general focus on judgement heuristics, including a strong focus on how the heuristics make us «smart», i.e., situations in which simple heuristics lead to accurate and useful judgements. Another difference between these two research approaches is that the ABC Group typically gives a more algorithmic description of the heuristics compared with the more informal descriptions of the Tversky-Kahneman tradition.

3. Heuristics and Biases

The three main heuristics described in (Kahnemann *et al.*, 1982) are (1) the representativeness heuristic, (2) the availability heuristic, and (3) the adjustment and anchoring heuristic.

3.1.Representativeness

The representativeness heuristic describes a judgement process based on the assumption that similarity with respect to some properties means similarity with respect to other properties. Tversky and Kahneman make a distinction between four basic cases of representativeness. The following case descriptions are adapted from Tversky & Kahneman (1982):

1) *M is a class and X is a value of a variable defined in this class*. The most representative value will be close to the mean, median or mode of the distribution of the relevant variable in the class M. For example, the most representative family income (X) may be the mean family income of the class of families (M). Representativeness is mainly determined in this case by what the judge knows about the frequency distribution of the relevant variable.

2) *M is a class and X is an instance of that class.* There are two ways in which an element can be highly representative of a class. An element is highly representative of a class if it is typical or if it is an ideal type that embodies the essence of that class. For example, if we know a person that we believe is a typical industry worker (X), we may use the characteristics of that person to represent characteristics of a class of industry workers (M).

3) *M is a class and X is a subset of M.* The criteria of representativeness are not the same for a subset and for a single instance because an instance can only represent the central tendency of attributes, whereas a subset can also represent range and variability. Similar to the previous example, we may know a group of industry workers (X), and use the characteristics of that group, e.g., the variance in opinions on a subject, to represent characteristics of a larger class of industry workers (M).

4) *M is a (causal) system and X is a (possible) consequence.* Here, X is representative of M either because it is frequently associated with M or because people believe, correctly or incorrectly, that M causes X. For example, a high inflation (X) can be representative for certain types of economies (M).

Tversky & Kahneman (1982) claim that most of the available research studies support the hypothesis that intuitive predictions and probability judgements are highly sensitive to representativeness, although not completely dominated by it. A strong reliance to representativeness works well in most cases, similarity with respect to one property frequently means similarity with respect to other properties. In addition, reliance of representativeness simplifies the judgement

[1] For more information about the ABC group, see http://www.mpib-berlin.mpg.de/abc/.

compared with the use of statistical, probability based, models. On the other hand, the use of the representativeness heuristic can be a major source of error, as we show in Section 5.1.

3.2.Availability

The availability heuristic is based on the assumption 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 (Tversky & Kahneman, 1974). Usually, this heuristic works quite well, common events are more available 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. Biases due to this heuristic may occur due to:

- *Own contribution*. The availability of experience increases with the level of participation in the actions leading to the experience (Ross & Sicoly, 1979).
- *The retrievability of instances*. For example, words starting with *r* are easier to retrieve than words with *r* as its third letter. Most people predict that words starting with *r* are more frequent, although the opposite is true (Tversky & Kahneman, 1974).
- *The imaginability of instance*. For example, if the potential problems of a software project are difficult to imagine, we will easily underestimate the risk. We will tend to do this even when we know from previous projects that there are many problems that we do not imagine at the start of a project. On the other hand, if we easily imagine horrible scenarios, e.g., death by drowning, a low likelihood of these scenarios may still lead to an overestimation of the risk (Tversky & Kahneman, 1974).
- *Illusory correlation*. For example, if we strongly believe that we can trust our intuition of when a software project will be a failure, confirming observations will be easier to recall than non-confirming observations. To scientifically test the quality of our intuition of project failure, we need to study *all* variants of the events A and B, where A = We felt that the project would be a failure, and B = The project was a failure, i.e., the four situations: (i) A and B, (ii) A and not(B), (iii) not(A) and B, (iv) not(A) and not(B). Assume that we have not studied situation (iii), i.e., the situation where we did not feel that the project would be a failure and the project was a failure. Then, a situation where there frequently are project failures and we, now and then, feel that a project will be a failure may easily lead to an illusionary belief in our predictions of project failures. Our intuition about project failure is, normally, not documented, and the we-knew-it-all-along bias may further increase the risk of a too high belief in our intuitions.

The availability heuristic has recently been challenged by some researchers, for example, (Sedlmeier *et al.*, 1998). A major problem with the evaluation of that heuristic seems to be the interpretation and measurement of the rather vague label «availability». In addition, there may be other mechanisms that can explain the judgement biases, e.g., the letter frequency biases, better than the availability heuristic.

3.3. Anchoring and Adjustment

The anchoring-and-adjustment heuristic consists of two steps:

1) Make an initial judgement, i.e., an anchor.
2) Adjust the initial judgement for differences between current situation and the anchor situation.

Studies by Tversky & Kahneman (1974) and Northcraft & Neale (1987) indicate that the adjustments are, most of the time, insufficient and that the anchor has an unexpected high impact on the final judgement. This can be illustrated by an example. In an experiment, described in (Kahnemann *et al.*, 1982) subjects were asked to estimate the percentage of African countries in the United Nations (UN). Before estimating, the subjects watched a wheel of fortune randomly stop at a number between 0 and 100. The subjects were instructed to indicate first whether that number was higher or lower than the percentage of African countries in UN, and then to estimate the percentage.

The randomly generated number had an amazingly large impact on the estimated percentage. For example, the median estimates of the percentage of African countries in the UN were respectively 25 and 45 for groups that received 10 and 65 as starting points!

4. Simple Heuristics that Make Us Smart

The heuristics described in (Gigerenzer *et al.*, 1999) are labelled «Fast and Frugal» heuristic, i.e., heuristics that are fast and computationally simple. Necessary conditions for a heuristic to be fast and frugal are that it is:

- sufficiently simple to be used by people without computational support and with realistic speed,
- specified algorithmically and includes rules for search, stop and decision making, and
- «ecological rational», i.e., the heuristic should exploit the structure of realistic environments and result in robust decision and learning models.

Heuristics satisfying these criteria should be evaluated with respect to, amongst others: performance compared with computationally more expensive strategies, e.g., regression based models, and the likelihood that people use these heuristics when learning and making judgements. If a fast and frugal heuristic performs well, and people do *not* use it, we may train people in when and how to use it. For this reason, an additional aspect is whether we can train people in using this heuristic correctly.

This section presents fast and frugal heuristics of the following classes:

- ignorance based heuristics,
- one-reason based heuristics, and
- elimination based heuristics.

4.1. Ignorance

Ignorance based heuristics are based on the fact that there is implicit information in the failure to recognise something, i.e., there is information in the fact that we have no information about an object. An example of ignorance based heuristics is the *recognition* heuristic: If one of two objects is recognised and the other is not, then it is inferred that the recognised object has the higher value (Gigerenzer *et al.*, 1999). A higher value means, for example, that the object is larger or better. The recognition heuristic is interesting for several reasons:

- It is very simple, only based on our ability to recognise objects.
- It is useful in situations where we have no other information than recognition and need a fast and frugal judgement.
- There is evidence that in some situations a person using this heuristic will perform better than a person using a more sophisticated heuristics and knowing more about the objects. In an experiment, American students performed better in predicting the larger of two German cites than the larger of two American cities. Gigerenzer *et al.* (1999) argue that this result is based on the fact that the students knew fewer German cities thus benefited from a strategy where they exploited the assumption that unknown cities were more likely to be smaller than recognised cities.

4.2. One Reason

One-reason based heuristics use only a single piece of information to decide between two alternatives, i.e., no combination of variable values is used to reach a decision. The three one-reason based heuristics described in (Gigerenzer *et al.*, 1999), (1) minimalist, (2) take the best, and (3) take the last, differ only in how they search for information. The perhaps most interesting of these heuristics is the Take The Best heuristic (TTB). TTB is used to select the object with the highest criterion value, e.g., the largest city, among a number of objects. TTB bases the selection on binary indicators, e.g., whether the city has a railway station. The indicator values of the objects are then compared pair-wise, starting with the most valid indicator. Indicator validity is for this heuristic defined as R/(R+W), where R is the number of observations where the object with the highest value of the indicator has the highest criterion value. For example, R equals the number of observations you have made where a city *with* a railway station is larger than a city *without* a railway station. W is the number of observations where this is not the case. The following example illustrates the use of indicator validity in TTB.

Assume that team A meets team B in a football match and that we want to predict whether A will win, i.e., our criterion is the outcome of the match. We have the two binary indicators with corresponding possible indicator values: I1 = {the team won the last match, the team did *not* win the last match}, and I2 = {the team is placed in the top 10% in the league, the team is *not* placed in the top 10% in the league}. We have 10 earlier observations connecting indicator and criterion values. In 6 (R) out of 10 (R+W) matches a win was followed by a win, and in 8 (R) out of 10 (R+W) matches the top 10% team won the match. Consequently, the validity of indicator I1 is 6/(6+4) = 0.6 and the validity of indicator I2 is 8/(8+2) = 0.8, i.e., indicator I2 is more valid than indicator I1.

TTB can be described in the following steps:

Step 0 (Recognition): If applicable, use the recognition heuristic; if neither of the alternatives are recognised, then guess; if both are recognised, go to Step 1.

For example, if we recognise only team A, we should according to the recognition heuristic predict that team A wins.

Step 1 (Search): In an ordered search choose the indicator with the highest validity that has not yet been tried for the choice selection process; then look up the indicator values of the two alternatives.

In the football match example, this means that we should start our search investigating the indicator values of I2 for the teams A and B.

Step 2 (Stopping rule): If one alternative has a positive value for the chosen indicator, while the other alternative has a negative or unknown value, then stop search and go to Step 3; otherwise go back to Step 1 and search for another indicator; if no further indicator is found, then guess.

If football team A is among the 10% best in the league and team B is not, then we can stop our search. If A and B have the same indicator value, then we investigate (Step 1) and compare (Step 2) the values of indicator I1.

Step 3 (Decision rule): Predict that the alternative with the higher positive indicator value has the higher value on the criterion. Assuming that both team A and B belong to the top 10% and that only team A won its last match, we should predict that team A will win against B.

Few studies are reported that analyse whether people really use TTB. However, the performance of this heuristic has been compared with regression analysis and other formal decision models in a number of tasks (Gigerenzer *et al.*, 1999). In most tasks TTB, in spite of its one-reason based decisions, performed as well or better than the competing formal decision models.

4.3. Elimination

Elimination-based heuristics extend one-reason based heuristics to a broader class of applications. While one-reason based heuristics are appropriate for *selection* between options, elimination-based heuristics are also appropriate for *classification* and *quantitative prediction* purposes. In our description of elimination-based heuristics below we explain the QuickEst heuristic and the categorisation by elimination heuristic as presented by Gigerenzer *et al.* (1999).

4.3.1. QuickEst

QuickEst is used for quantitative prediction purposes. It assumes that an object has a set of variables and that we want to predict the value of one of these variables. The variable value to be predicted is, as before, called the criterion value and the object variables, when used for prediction purposes, are called indicators. QuickEst is based on, similar to the Take The Best heuristic, binary indicators. Here we label the two binary indicator values using a '-' and a '+' sign. Further, we define s_i^- as the average criterion value for the objects with the value '-' on the indicator *i* and s_i^+ the other value. To simplify the algorithmic description we consequently use ‘-‘ on the values with the lowest criterion value, i.e., s_i^- is always less or equal than s_i^+. We define that the validity of indicator *i* is higher than the validity of indicator *j* if s_i^- is *lower* than s_j^-.

The prediction process starts with a comparison of values of the indicator with the highest validity, i.e., a comparison of the value of the most valid indicator of the object to be predicted with the ‘-‘ value of the same indicator. If these values are equal, we stop the search and use the value s_i^- as our predicted criterion value. If not, we proceed with less and less valid indicators until we find equal values, if no equal values are found, we use the highest s_i^- value. Note that starting with the *lowest* values leads to faster decision, compared with starting with the highest values, if the criterion values belong to a distribution with many small values and few high values. If this is not the case, other starting points may be better.

In spite of the perhaps complicated description above, QuickEst itself is simple. Assume, for example, that we want to predict the number of medals won by Morocco in the Sydney Summer Olympics and that we will use QuickEst. Most countries win very few medals and we start investigating indicators that imply that Morocco won no medals. One such indicator is, we may believe, that small African countries seldom win medals in the Olympics. Morocco is, however, not a *small* African country. Moreover, we do not find other indicators that imply no medals. Therefore, we continue with indicators implying that Morocco won, for example, 1 to 3 medals, 4 to 8 medals, etc. until we find the first true indicator. If the first true indicator with corresponding average criterion value is that we believe that «African countries with 10 to 30 million inhabitants win on average 4 to 8 medals in the Summer Olympics», our prediction is that Morocco won 4 to 8 medals. Note that the indicators in this example are not based on actual indicators, but on what we might believe. It is, of course, possible to improve the use QuickEst with actual historical indicator values.

An ongoing experiment carried out by the authors instructed the subjects to predict the number of medals won by Morocco and other countries in the Sydney Olympics. The analysis indicates, so far, that the subjects used strategies similar to QuickEst when the knowledge about the actual medals won was poor. Surprisingly, the subjects having more knowledge about the actual distribution of medals won in the Summer Olympics did not predict better than the subjects using a QuickEst similar strategy.

4.3.2. Categorisation

Categorisation by elimination is a heuristic used to predict which category an object belongs to. Similar to the earlier heuristics, we rank indicators according to validity. The most valid indicator in this heuristic is the indicator that, when used alone, make most correct category predictions. For example, assume that we have parked our car in a car parking, been away for a while and now want to find the car. We do remember the colour (red), the type of the car (Volvo) and the area where we parked it (slot B1). The strategy we may use for finding our car is to eliminate all cars that cannot be ours, i.e., to determine whether a car belongs to the category «not our car». To categorise cars into «our car» and «not our car», we may use the indicators car colour, car type and car parking area. Assume that we believe that the indicator values in the car parking are distributed as shown in Table 2.

Indicator values	**Red**	**NOT(red)**	**Volvo**	**NOT(Volvo)**	**B1**	**NOT(B1)**
Our car	1	0	1	0	1	0
Not our car	19	80	14	85	9	90

Table 2: Indicator value distribution

From the indicator value distribution we can calculate that the indicator car colour predicts correctly in 81% of the cases, car type 86% and car parking area 91% of the cases. The most valid indicator is, therefore, car parking area, then, car type and, finally, car colour. In a realistic situation we would, probably, modify this validity measure to take the *effort* we need to identify the different indicator values into account.

The categorisation by elimination heuristic follows the following algorithm:

1) Let S be the whole set of categories. In the car parking example, S = {Our car, Not our car}
2) Get next indicator I. Indicators are ordered with respect to categorisation validity. The most valid indicator in our example is the car parking area, i.e., we start with I = car parking area.
3) Set possible categories P to categories corresponding to indicator I's value. Assume that the car we attempt to categorise is a *red Saab*, parked in *B1*. Starting with the most valid indicator, P is set to {Our car, Not our car} because both our car and other cars are parked in B1.
4) a) If S ∩ P contains only one category, then select this category. End.
   b) If S ∩ P contains more than one category, let S = S ∩ P, go to 5)
   c) If S ∩ P = ∅, go to 5).
   In our example, the intersection between S and P contains two categories, i.e., we proceed with Step 5.
5) a) If there are no more indicators available, select a category from S at random.
   b) If there are more indicators available, go to 2).
   In our example, there are more indicators available, car type and car colour, and we proceed with Step 2 and start the second iteration of the algorithm.

In the second iteration in the car parking example we use the second most valid indicator; I = car type. P is now set to {Not our car} since the car we observe is a Saab and our car is a Volvo. This means that the category Our car is eliminated. Hence, the intersection between S and P is {Not our car}. According to step 4 we predict that the car is not our car and end the categorisation of that car.

In (Gigerenzer *et al.*, 1999) this categorisation heuristic was applied to categorise flowers, wine and mushrooms. The simple heuristic performed almost as good as formal statistical methods on these data sets. In situations where there is a degree of uncertainty regarding the indicator values of the object we want to categorise, the heuristic may however be difficult to use. For example, assume that we were only 80% sure that we had parked our car in B1 and that we were unable to exclude any parking area completely. The validity of the parking area indicator would still be high, because the use of it leads to a high proportion of correct categorisations. On the other hand, observing a car in B5 we would be almost sure that the car is not ours, but the original heuristic would not allow us to use this information to exclude the category Our car. In other words, there may be few situations where we actually use categorisation by elimination as described in this heuristic.

5. The Use of Heuristics in Software Processes

5.1. Representativeness

Judgements relying on the representativeness heuristic, typically, search for the most representative elements from a class of elements. For example, a software development risk analysis of project A based on representativeness builds upon on a search for the most similar projects from a class of completed software projects. Then, the similar projects are used to draw conclusions about the risk of project A. Consequently, the heuristic may lead to poor judgement when more similar is not the same as more useful for prediction purposes.

The following more detailed example of high representativeness and relatively low prediction accuracy is described in (Jørgensen & Sjøberg, 2001b). This study indicates that software maintainers frequently use the representativeness heuristic together with the anchoring-and-adjustment heuristic when they carry out risk analyses of maintenance tasks. The maintainers tried to remember the most representative tasks carried out recently and adjusted their risk assessment due to differences between the representative and the new task. Although the maintainers had detailed knowledge about the task they was supposed to complete, a simple judgement model based solely on whether the size of the task was small or medium/large made more accurate risk assessments than representativeness based assessments of the maintainers. Similar results on how a strong reliance of representativeness may lead to poor judgements are reported in, for example, (Dawes & Corrigan, 1974).

To avoid the misuse of the representativeness heuristic, we must understand its limitations. For example, consider a new software project A, which is very similar to a completed project B. Project B had an extremely high productivity. This extremely high productivity is, however, not only caused by the variables that make project A similar to project B, but also by other variables, such as an unusual degree of luck. Therefore, it is unlikely that project A will have as high productivity as project B. This is the statistical *regression toward the mean* effect, i.e., the same effect that explains why it is likely that children of tall parents will be smaller than their parents (Campbell & Kenny, 1999), (Jørgensen *et al.*, 2001). Due to this effect it would have been better to use less similar projects, i.e., more average projects, as a basis for the estimation of the productivity of project A.

Software risk assessment and productivity predictions should be based on knowledge about when people tend to think too representatively. In these cases more formal data analysis may be necessary for accurate judgements. We further evaluate the use of the representativeness heuristic when estimating software project productivity in Section 6.

5.2. Availability

The availability heuristic aims at describing how people make judgements about frequency and probability. According to this heuristic, human judgements on frequency and probability are based on how easily they can recall objects or events. Objects or events that can easily be recalled are assumed to be more frequent than objects or events that are more difficult to recall. This heuristic may explain why we found that software maintenance managers assessed the proportion of effort spent by their maintenance team on enhancing and adapting software applications reasonably accurate, while the proportion of effort spent on error correction was highly over-estimated (Jørgensen, 1995b). Software corrections may receive more management attention compared with other types of maintenance work and are, therefore, easily overestimated according to the availability heuristic.

Knowledge about when the availability heuristic leads to biased judgements is an important input when a decision has to be taken on whether to trust frequency judgements based on expert judgement or start a measurement program to collect and analyse data more objectively. If there are reasons to believe that there are significant differences in how easily people recall different events, more objective measurement may be necessary to get accurate judgements.

5.3. Anchoring and Adjustment

The anchoring-and-adjustment heuristic is a two step judgement process. First, an initial judgement is made and its result is established as the anchor. Then, adjustments are made for differences between the current situation and the anchor situation. There are indications that the anchoring-and-adjustment heuristic is important when experts estimate software project effort and, sometimes, a significant reason for estimation inaccuracy (Jørgensen & Sjøberg, 2001a). Such indications were found both in industrial contexts using project experience reports and interviews as sources, and in an anchoring-and-adjustment experiment performed with 38 computer science students. Here, we will illustrate this experiment in more detail:

- The students estimated how much effort they would use on a course in software engineering, i.e., they had to deliver an estimate of the effort on own work. The estimation was carried out approximately 4 weeks after the course had started. The activities included in the estimate were: participation in the lectures, participation in the weekly exercises, work on the mandatory software development project and self-study on course topics. The students could use the activity category «Other activities» if an activity did not fit any of the activities above.
- Before estimating the course effort in detail the students were asked whether they thought they would use more than X work-hours on the course. Half of the students were presented with X=200 work-hours and the other half with X=800 work-hours. While 200 work-hours was a possible effort usage, 800 work-hours would mean that the students used about 50 hours each week on the course, i.e., a much too high effort estimate. The students were asked

not to use more than 5 minutes on this initial estimation task and were informed that the X-value was not based on historical data or what the course teachers believed was a reasonable amount of effort. We believed that the X-value would, in spite of this information, be used as an anchor and would have an impact on the activity estimates.

- Then, the students were asked to give an effort estimate on each of the activities of the course. They were asked to estimate the minimum, average and maximum effort on each activity.
- The average estimated effort for groups with different X-values were finally compared.

While the group presented with X=200, on average, estimated 202 work-hours, the group presented with X=800 estimated, on average, 303 work-hours, see Figure 1.

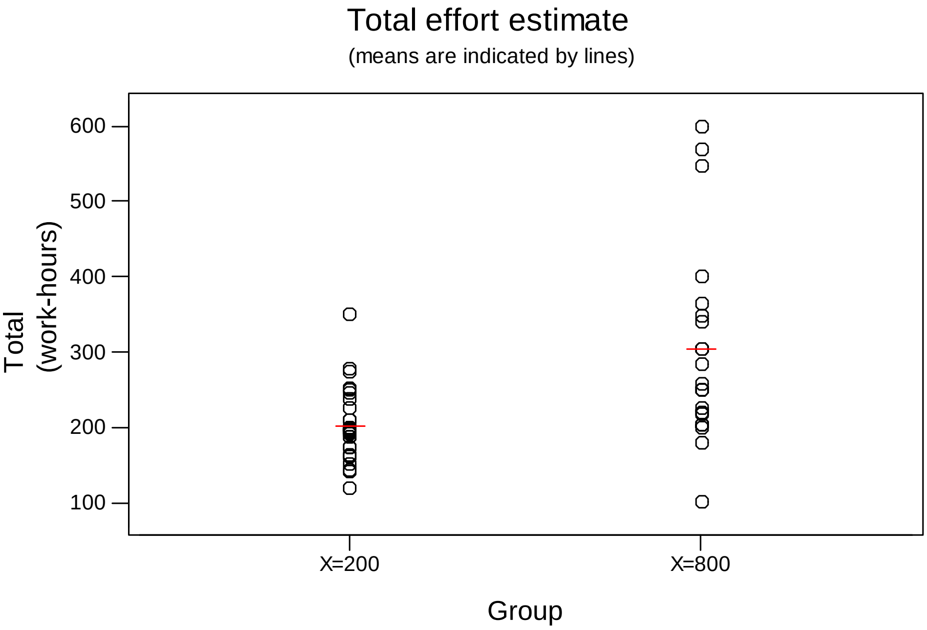

Figure 1: Anchor and adjustment experiment

A statistical one-sided t-test with different variance gave p=0.003, i.e., the difference is significant. We asked the students whether they believed their estimate had been impacted by the X-value. In spite of a somewhat leading question: «Do you believe that you have been influenced ...», less than half of the students believed that they had been influenced. An analysis of the data showed no significant difference in estimates between those that believed that they had been impacted and the others. This indicates that people are not always aware of the anchors used and their impact on the judgements. More details about this study can be found in (Jørgensen & Sjøberg, 2001a).

When estimating software work it is, according to the results above, important that the estimation is carried out independently of non-informative or biased numbers. For example, if the software product customer has unrealistic expectations regarding the cost of a project, estimators aiming at *realistic* estimates should not know these expectations! As indicated in the experiment the estimators may otherwise believe that they have provided a realistic effort estimate, when in fact they have provided a much too optimistic estimate anchored in the customer expectations.

Similar results were found by Hoch & Schkade (1996), who reported that an estimation process based on anchoring-and-adjustment performed well compared with other estimation methods when the anchor projects were good predictors of the new project, and performed poorly when the anchor projects were different from the new project, i.e., there was an insufficient adjustment for the differences. Interestingly, the knowledge of the human characteristic that the differences to closest analogy is not sufficiently adjusted for, can be used to compensate for another human weakness, namely the tendency of planners to make too optimistic predictions. The compensation technique, used by some project leaders, is to start the estimation process with the most pessimistic effort estimate and then estimate the most likely and the most optimistic effort. The most pessimistic effort estimates will then function as the anchor values and lead to higher estimates of most likely effort compared with an estimation process starting with the most likely effort estimates. Results supporting this compensating process are reported by Conolly & Dean (1997).

5.4. Ignorance: Recognition

The recognition heuristic is based on the assumption that people, when choosing between two objects, prefer the object they know if they do not know the other object. In software process improvement work, the recognition heuristic may, for example, be used together with other heuristics when selecting software development tools. There is, typically, no empirical evaluation of the overall performance of a development tool to support such selections (Jørgensen *et al.*, 1995). An important input in the selection process may, for this reason, be the *strength* of recognition of the tools. This example is based on an extension of the recognition algorithm described in Section 4.1.

While the original recognition heuristic divides recognition into recognised and not recognised, our example is based on a more fine-grained scale of recognition. This extension is, in our opinion, a natural extension, but lacks the preciseness of the original recognition heuristic since there is no commonly accepted fine-grained scale for strength of recognition. The strength of development tool recognition is increased with information about other companies using the tool, tool advertisements, vendor presentations and other tool marketing initiatives. In our opinion, the recognition heuristic explains why many of the advertisements do not focus on *informing* the potential customers about the performance of the tool. Instead, they focus on increasing the strength of recognition. Strength of recognition may correlate with important performance characteristics and may be a useful strategy if a fast decision is needed. Selecting less recognised development tools may, for example, imply smaller tool vendors with high risk of no further development of the tool. On the other hand, if our intuition, which many people may trust more than analytical models, is too much based on strength of recognition, a vendor with a poor development tool can compensate for this through demonstrations and advertisements. The recognition heuristic can, thus, when used alone, lead to poor decisions.

5.5. One Reason: Take The Best

A decision process based on the heuristic Take The Best (TTB) investigates indicators according to their validity and stops when a difference between two alternatives is found. Take the following example, a project leader has decided to improve the project performance by introducing a code generating CASE (Computer Aided Software Engineering) tool. The project leader has collected information about two CASE tools, A and B, and has to choose one of them. Here, the original recognition heuristic cannot be used, since both A and B are recognised. Assume that the main criterion when choosing a CASE tool is project productivity and that the most valid indicators of productivity are (1) quality of generated code and (2) ease of CASE tool use. The project leader sorts the indicators according to their validity, i.e., how useful the indicators are believed to differentiate between the CASE tools. The search for differences starts with the indicator with the highest validity, and, if there are no significant differences between A and B concerning this indicator, continues with the second most valid indicator, and so on, until a difference is found. When a difference is found, the CASE tool with the highest value for this indicator is chosen. Note that TTB does not combine indicators and that less valid information is not considered at all if a significant difference on a more valid indicator is found.

When does TTB perform well, and is this heuristic actually used by people? Experiments described by Gigerenzer (1999) indicate that TTB potentially explains elements of human judgement and performs just as well as or better than more formal methods, e.g., regression based models and classification trees, when:

- the number of objects per indicator is low, i.e., when there are many valid indicators and few observations to learn from, and

- the information is relatively non-compensatory, i.e., the indicator validity is strongly decreasing. Information is totally non-compensatory when each indicator is more valid than any combination of less valid indicators.

The former condition may be typical for software improvement decisions. For example, in the CASE tool example above we may have only few observations that indicate the performance of the CASE tools and many indicators that are relevant for the performance of the tools. The second condition is less obvious. TTB will perform well if there is a strong decrease in indicator validity. This is the case if, for example, quality of CASE tool generated code is considered the most important characteristic and only very large differences regarding ease of use should alter a decision based on a difference in quality of generated code. On the other hand, TTB performs badly if there is important information in a combination of indicators, e.g., when weighting of indicators is necessary to make a good decision. We recommend the use of formalised decision procedures when there is no clear ranking of indicator validity or when the combination of indicator values are believed to contain important information. This recommendation is in accordance with the results described in (Dawes, 1988), where it is reported that people, on average, perform poorly in situations where two or more aspects of a situation have to be attended at once, i.e., situations where combination of indicators are important. However, if there is a clear ranking of indicator validity, i.e., the indicator validity is strongly decreasing, and there are few relevant observations available, a decision based on expert judgement may perform just as well or better than a formalised decision procedure.

5.6. Elimination: QuickEst

In the following example, we estimate the software productivity measured as function points (Symons, 1991) per work-hour of a new project using QuickEst. The example use a data set of real life, completed, software projects described in (Jeffery & Stathis, 1996). The productivity indicators and the corresponding indicator values are listed in Table 3.

Indicator	Indicator values
Programming language	COBOL, C, 4GL, mixed
Project type	Distributed platform, centralised platform
Project size	Small, medium, large

Table 3: Indicators and indicator values

For each indicator value we calculated the average productivity and ranked them according to this value. The indicator value with the lowest average productivity value was allocated the highest validity rank, the second lowest was allocated the second highest validity rank, etc. Table 4 shows the 5 most valid indicator values.

Rank	Indicator values	Average productivity value for indicator value
1	Programming language = mixed	0.10 function points per work-hour
2	Project type = distributed	0.13 function points per work-hour
3	Programming language = C	0.21 function points per work-hour
4	Programming language = COBOL	0.22 function points per work-hour
5	Programming language = 4GL	0.25 function points per work-hour

Table 4: Validity ranked indicator values

There were additional indicator values in the data set, e.g., Project size = large, but using QuickEst on our data set, no other than the above indicator values had to be applied to estimate the project productivity. For example, a project using 4GL as programming language on a distributed platform will, regardless of project size, result in a QuickEst productivity estimate of 0.13 function points per work-hour in only two search steps. The prediction accuracy, defined as the mean value of the difference between the actual and the QuickEst predicted productivity relative to the actual productivity, was 38%. Interestingly, neither estimation by analogy using clustering algorithms based on Euclidian distance nor linear regression achieved a better prediction accuracy than QuickEst. Just as important, while the statistical models frequently over-fitted the model to the data, e.g., developed a prediction model modelling the productivity of projects in the learning set very well on the cost of the prediction of new projects, QuickEst avoided the over-fitting to a large extent.

Independently of whether expert estimators actually use a heuristic similar to QuickEst, this heuristic illustrates that fast and frugal heuristics can have a performance comparable to more sophisticated methods. QuickEst enables probably better integration with expert estimator knowledge than formal estimation models because the heuristic is easy to understand and the indicator values are easy to update with new experience. In addition, it is easy to train the software professionals in using QuickEst when estimating the productivity, effort or time usage of new projects.

5.7. Elimination: Categorisation

We evaluated the categorisation by elimination heuristic on the following task: Categorise software maintenance tasks to be completed into the two categories: (C1) the task will have major unexpected problems and (C2) the task will not have major unexpected problems. Our evaluation was based on a data set of 109 maintenance tasks described in (Jørgensen, 1995a). The most valid indicators are described and ranked according to percentage of correct predictions in Table 5, as described in Section 4.3.2.

Rank	Indicator	Correctly predicted categories and prediction rule
1	Size of change	62 % correct predictions. IF Size of change = small THEN predict C2 ELSE predict C1
2	Type of change	58% correct predictions. IF Type of change = corrections THEN predict C1 ELSE predict C2
3	Confidence of maintainer	57% correct predictions. IF Confidence of the maintainer = high THEN predict C2 ELSE predict C1 The confidence of the maintainer is high if he/she believes to know how to solve the task when starting the maintenance task work.

Table 5: Categorisation indicators sorted by validity

Unfortunately, none of these indicators eliminate categories. There are small tasks that have major unexpected problems, there are corrective tasks without major unexpected problems and there are tasks where the maintainers' confidence are high that have major unexpected problems. Following the algorithm described in Section 4.3.2, we should then select category (C1) or (C2) at random. The original categorisation heuristics based on elimination may, therefore, not be very useful in situations like this, where we are unable to eliminate any alternatives based on the available information. In the above case, there is an obvious lack of fit between the heuristic and its environment. Hence, the heuristic Take The Best, for example, may be a better choice.

The main purpose of this example is to illustrate that to perform well, software process experts need a toolbox of heuristics, and knowledge about when there is a fit between the heuristics and their environments.

6. Experiment on Use of Representativeness

Our hypothesis is that human judgement heuristics differ from person to person based on, amongst others, a person's toolbox of heuristics and previous feedback on the usefulness of a heuristic. To illustrate the variety of how a judgement heuristic may be used, this section describes how individuals differ in their use of the representativeness heuristic when estimating software development productivity.

We carried out an experiment including 36 under-graduate students who followed a course in software engineering at the University of Oslo. The use of students instead of software professionals may be a threat to the validity of the results of this experiment. However, interviews with professional software developers indicate that they, typically, get no formal training in estimation and that the feedback of the estimates they make is poor (Jørgensen & Sjøberg, 2001a). In addition, we believe that our software engineering students can be considered as semi-professionals. They practise software development as part of the university courses, and most of them work parttime in the software industry. For this reason, we believe that there is no large difference between how students and software professionals estimate software development projects although they might differ in the amount of experience. Our view is supported by Höst *et al.* (2000), who found that computer science students were good substitutes for software professionals to assess lead-time impact factors. The students in our experiment were asked to estimate the productivity of a new project (P-NEW). They were presented with information about P-NEW and the results of a selection of the two most similar projects, called P-11 and P-34, from a project database, see Table 6.

Projects	Platform	Architecture	Size	Complexity	Productivity
P-11	Cobol/MVS	Client/server	9 kLOC	HIGH	0.3 kLOC/PM
P-34	Cobol/MVS	Client/server	11 kLOC	HIGH	1.1 kLOC/PM
P-NEW	Cobol/MVS	Client/server	10 kLOC (estimated by the project leader)	HIGH (estimated by the project leader)	? LOC/PM

Table 6: Project information used in the experiment (kLOC = 1000 Lines of Code, PM = Person Months)

In addition to the information in Table 6, the following information about the projects in the project database was presented:

- The project database consists of 100 software development projects.
- The projects varies in productivity from 0.2 kLOC/PM to 7 kLOC/PM.
- The average productivity of all 100 projects is 2 kLOC/PM.
- The average productivity of the ten projects with development platform and complexity identical to P-NEW is 1.0 kLOC/PM. (These are the two factors considered most important for the productivity.)

Then, the students were asked: What would you, based on the available information, suggest as productivity estimate for P-NEW? Explain how you derived this estimate.

When we analysed and classified the strategy of each estimate, both the estimate itself and the explanation were taken into account. The resulting answers were classified into 5 strategies of use of the representativeness heuristic, depending on the projects used as analogues to predict the productivity of P-NEW:

1) Use of the two closest analogues (P-11 and P-34)
2) Use of the average of the ten closest analogues and adjustments for P-11
3) Unadjusted use of the average of the ten closest analogues
4) Use of the average of the ten closest analogues and an adjustment for the total average

Table 7 shows the distribution of the strategies and the average estimated productivity for each of them.

Strategy	Frequency	Average estimate
1	9 (24%)	0.8 kLOC/PM
2	7 (18%)	0.9 kLOC/PM
3	17 (45%)	1.0 kLOC/PM
4	5 (13%)	1.7 kLOC/PM

Table 7: Estimation strategies

Although these results need to be replicated in more realistic contexts, we have an indication that the actual use of the representativeness heuristic is based on very different individual interpretations of representativeness. In our case of the estimation task these were, for example:

- P-11 is representative for P-NEW because P-11 is similar to P-NEW

- P-11 is *not* representative for P-NEW because P-11 has an unusually low productivity
- The average of the ten closest analogues is representative for P-NEW, with the probable reasoning that the variance of possible productivity values cannot be represented by one or two earlier projects.

The expected prediction accuracy of the above strategies depends on how well the use of the project information in the experience database enables us to predict new projects. If high similarity means high predictability, then the use of few analogues to predict a new project may be a proper strategy. On the other side, if the project information in the experience database is of less use to predict new projects, then use of the *total* average productivity may be the best estimation strategy. Information about the predictability of the historical data was not included in the task description. None of the students made, however, any assumptions on the predictability of the data in the experience database when answering the estimation question. The students may have assumed environments with different estimation uncertainty when selecting their estimation strategy. The results do consequently not necessarily describe the variance of productivity estimates in a realistic environment, with better known estimation uncertainty. The results, however, indicate the following:

- To select a proper estimation strategy, the information about the estimation uncertainty is essential.
- The awareness of the relation between predictability and use of very similar projects as predication analogues can be low.
- The representativeness heuristic is a rather vague label for several different strategies that may lead to very different estimates.

As shown in Table 7, nine of the estimates were based on Strategy 1, i.e., the strategy that the closest two analogues were the best predictors. As described earlier, Strategy 1 is expected to perform poorly if the estimation uncertainty is low. To further investigate whether the students were aware of this relation between estimation strategy and uncertainty, we added a second question: Assume that the estimation accuracy of estimates based on the 2-3 most similar projects has, on average, been very low. Does this information have an impact on the productivity estimate? Explain why/why not?

We found that only one of the nine students using Strategy 1 to solve the first task changed the estimate based on this information. The other eight students answered that the uncertainty had no impact on their estimates. Their choice of strategy did, therefore, not change even to the somewhat leading question formulation above. This points at a need for training in the relation between prediction uncertainty and use of the representativeness heuristic. This type of improvement is an example of software improvement *supporting* the expert estimators instead of *replacing* the expert with statistical «black-box» estimation tools.

7. Conclusions and Further Work

This paper has exemplified that software processes can be improved applying knowledge about human judgement heuristics. The main condition for a human judgement heuristic to perform well is that there is a fit between the heuristics and their environment. For example, in environments where the recognised objects on average are better than the unrecognised objects and where there are scarce resources to collect information, a good strategy is to select between the initially recognised objects. There are studies reporting amazingly good judgements based on little information and simple computations when there is a fit between the judgement strategies and the environments. On the other hand, there are numerous examples of how simple heuristics may lead to poor decisions, in particular, when the estimators are unaware the heuristics they use and unconscious about their assumptions regarding the fit between the heuristic and the environment. A small-scale experiment carried out by the authors of this paper indicated that different persons' interpretation of the representativeness heuristic may differ very much and is not necessarily based on awareness of the assumptions behind this heuristic. In particular, software estimators may need training in the relation between representativeness and estimation uncertainty.

This paper has described initial research on how to improve software processes based on an understanding of how software professionals judge and learn from experience. In the near future we plan to carry out several industrial studies and experiments to study issues such as:

- Can elements of the Take The Best heuristic improve software effort estimation?
- How do professional software estimators use data from earlier projects?
- How typical is the faulty use of the heuristics applied by software professionals, e.g., do software professionals select an estimation strategy in accordance with the prediction uncertainty of their environment?
- When is there a major lack of environmental fit between the heuristic and its environment?
- How does the use of heuristics differ among different individuals and how can software professionals be trained to use the most proper heuristic?
- Will training of the software professionals based on knowledge about the heuristics actually improve software development processes?

Acknowledgements

The research project is funded by The Research Council of Norway through the industry-project PROFIT (PROcess improvement For the IT industry). Thanks to Karl Heinz Kautz who reviewed the paper and gave many very useful suggestions.

References:

Blattberg, R.C. & Hoch, S.J. (1990) Database models and managerial intuition: 50% model + 50% manager, *Management Science,* 36, pp. 887-899.

Briand, L.C., El Emam, K. & Bomarius, F. (1998) COBRA: A Hybrid Method for Software Cost Estimation, Benchmarking, and Risk Assessment, *Int. Conf. on Software Engineering,* Kyoto, Japan.

Brown, N.R. & Siegler, R.S. (1993) The role of availability in the estimation of national populations, *Memory and Cognition,* 20, pp. 406-412.

Campbell, D.T. & Kenny, D.A. (1999) *A primer on regression artifacts,* The Guilford Press.

Collins (1991) *Collins english dictionary,* Aylesbury, England, HarperCollins Publishers.

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

Conte, S.D., Dunsmore, H.E. & Shen, V.Y. (1986) *Software engineering metrics and models,* Benjamin/Cummings Publishing Company Inc.

Dawes, R.M. (1988) *Rational choice in an uncertain world,* Harcourt Brace & Company.

Dawes, R.M. & Corrigan, B. (1974) Linear models in decision making, *Psychological Bulletin,* 81(2), pp. 95-106.

Deming, W.E. (2000) *Out of the crisis,* MIT Press.

Finnie, G.R. & Wittig, G.E. (1997) A comparison of software effort estimation techniques: using function points with neural networks, case based reasoning and regression models., *J. Systems Software,* 39, pp. 281-289.

Gigerenzer, G. & Goldstein, D.G. (1996) Reasoning the fast and frugal way: Models of bounded rationality, *Psychological review,* 103, pp. 650-669.

Gigerenzer, G., Todd, P.M. & group, A.R. (1999) *Simple heuristics that make us smart,* New York, Oxford University Press.

Groner, M. (1983) *Methods of heuristics,* NJ, Erlbaum.

Hoch, S.J. & Schkade, D.A. (1996) A psychological approach to decision support systems, *Management Science,* 42(1), pp. 51-64.

Höst, M., Regnell, B. & Wohlin, C. (2000) Using students as subjects - a comparative study of students and professionals in lead-time impact assessment, *Empirical Software Engineering,* 5(3), pp. 201-214.

Jeffery, F.R. & Stathis, J. (1996) Function point sizing: structure, validity and applicability, *Empirical Software Engineering,* 1(1), pp. 11-30.

Jørgensen, M. (1995a) An empirical study of software maintenance tasks, *Journal of Software Maintenance,* 7, pp. 27-48.

Jørgensen, M. (1995b) The quality of questionnaire based software maintenance studies, *ACSM SIGSOFT - Software Engineering Notes,* 20(1), pp. 71-73.

Jørgensen, M. (1997) An empirical evaluation of the MkII FPA estimation model., *Norwegian Informatics Conference,* Voss, Norway.

Jørgensen, M., Bygdås, S.S. & Lunde, T. (1995) Evaluation of CASE Tool Efficiency - Method and Results, *Applied Software Measurement,* Orlando, USA.

Jørgensen, M., Indahl, U. & Sjøberg, D. (2001) Software effort estimation and regression toward the mean, *Submitted to: SEKE 2001*.

Jørgensen, M. & Sjøberg, D.I.K. (2001a) Impact of software effort estimation on software work, *Submitted to Journal of Information and Software Technology*.

Jørgensen, M. & Sjøberg, D.I.K. (2001b) Learning from experience in a software maintenance environment, *Submitted to IEEE Transactions on Software Engineering.*

Kahnemann, D., Slovic, P. & Tversky, A. (1982) *Judgement under uncertainty: Heuristics and biases,* Cambridge University Press.

Kemerer, C.F. (1987) An empirical validation of software cost estimation models, *Communications of the ACM,* 30(5), pp. 416-429.

Myrtveit, I. & Stensrud, E. (1999) A controlled experiment to assess the benefits of estimating with analogy and regression models, *IEEE Transactions on Software Engineering,* 25, pp. 510-525.

Northcraft, G.B. & Neale, M.A. (1987) Experts, amateurs, and real estate: An anchoring-and-adjustment perspective on property pricing decisions., *Organizational Behavior and Human Decision Processes,* 39, pp. 84-97.

Paulk, M.C., Curtis, B., Chrissis, M.B. & Weber, C.V. (1993) Capability maturity model, Version 1.1, *IEEE Software,* 10(4), pp. 636-651.

Plous, S. (1993) *The psychology of judgment and decision making,* McGraw-Hill.

Pruijt, H.D. (1997) *Job design and technology: Taylorism vs. Anti-Taylorism,* London, Routledge.

Ross, M. & Sicoly, F. (1979) Egocentric biases in availability and attribution, *The journal of personality and social psychology,* 37, pp. 322-336.

Sedlmeier, P., Hertwig, R. & Gigerenzer, G. (1998) Are judgements of the positional frequencies of letters systematically biased due to availability?, *Journal of Experimental Psychology: Learning, Memory and Cognition,* 24(3), pp. 754-770.

Symons, C. (1991) *Software sizing and estimating: MkII Function Point Analysis,* J. Wiley and Sons.

Taylor, F.W. (1911) *The principles of scientific management,* New York, Harper & Row.

Tversky, A. & Kahneman, D. (1974) Judgment under uncertainty: Heuristics and biases, *Science,* 185, pp. 1124-1130.

Tversky, A. & Kahneman, D. (1982) Judgments of and by representativeness, in: D. Kahneman, P. Slovic & A. Tversky (Eds) *Judgment under uncertainty: Heuristics and biases,* Cambridge, England, Cambridge University Press.