Predicting software development skill from effort predictions

(Un)skilled and unware of it?

Magne Jørgensen
SimulaMet & OsloMet

The Dunning-Kruger effect:

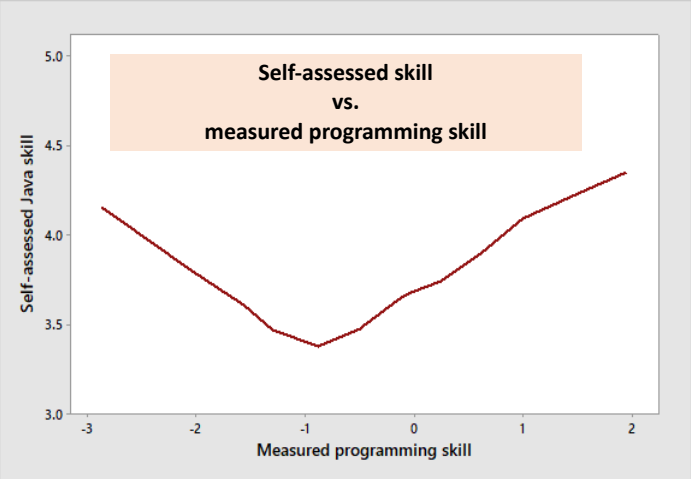
«Unskilled and Unaware of It: How Difficulties in Recognizing One's Own Incompetence Lead to Inflated Self-Assessments»

*Kruger, Justin; Dunning, David (1999). Journal of Personality and Social Psychology. **77** (6): 1121–1134.*

Our study

- 104 software programmers from two offshoring companies
- Collection of programming skill information
  - Information from their CVs
  - Company skill category (junior, intermediate, senior programmer)
  - Self-assessed programming skill
- Predicting the effort they believed they most likely would need to complete four larger and five smaller tasks (nine effort predictions)
  - Self-assessed skill in terms of effort needed to solve the tasks
- Completing a programming test, consisting of programming the five smaller tasks
  - Measured programming skill (based on skill assessments instruments developed and validated by Gunnar Bergersen in his PhD-theses)

Identifying the most competent software developers is difficult

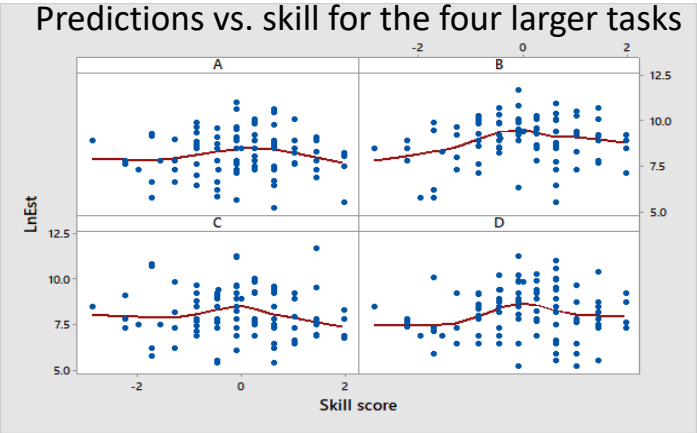


Other indicators were also poor:

- Length of experience
- Company skill category (junior, senior)
- Confidence in knowing how to solve the task

What to do?

Could we use their effort predictions as skill indicators?



Dunning-Kruger effect again:

Lowest effort predictions among those with the highest and lowest skill

What if we ask the developers to predict the effort of tasks that looks complex (especially for those less skilled), but are actually not that difficult
- i.e., our five smaller tasks

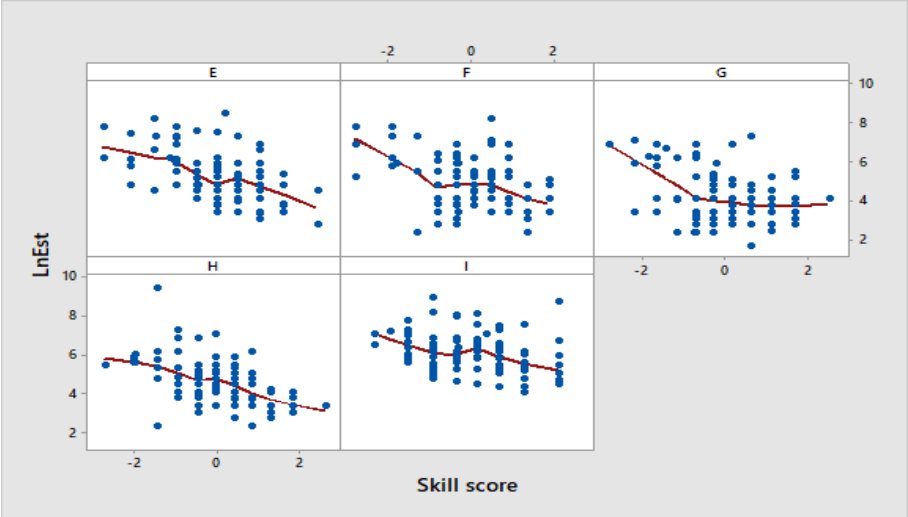
Would high and low effort prediction then separate those with low and high skill?
Would then the normal Dunning-Kruger effect be reversed?
- From «Unskilled, but unaware of it» to «Skilled, but unaware of it.»

Example of one of the smaller programming task: Coffee Machine

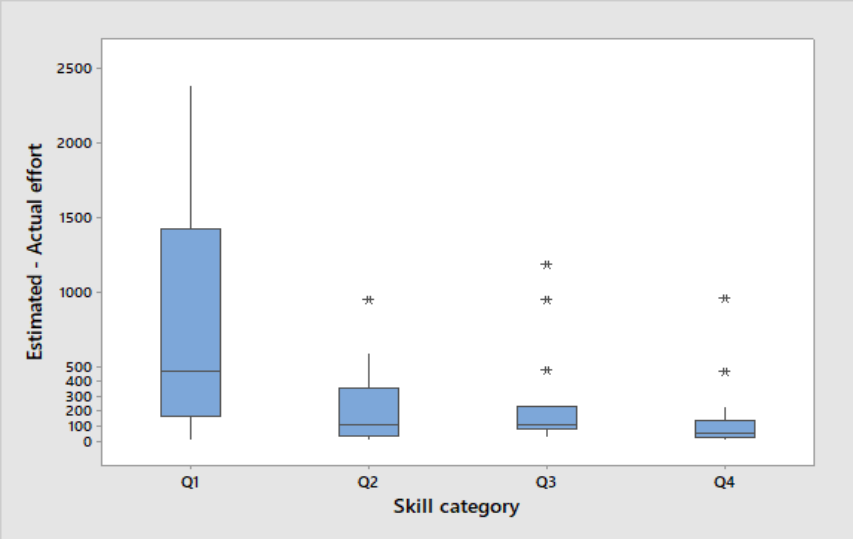
- Four pages of task description (including a design diagram in UML)
- 14 pages of programming code (in Java)

- BUT, very few lines of code to write and update in the code controlling the coffee machine. A developer, even in the lowest skill category, would usually only spend a few minutes to solve this task.

Estimates of smaller, seemingly complex, but actually rather easy tasks separate low and high skill



Over-pessimism (Predicted-Actual effort) Task F



Large, complex tasks (A..D): Predictions not useful as skill indicator
Smaller seemingly complex tasks (E..I): On the edge of being useful?

Task	Somers' D and hit rate	Correlation between estimate and skill
Task A	D = −0.05, hit = 47% (p = 0.22)	r = 0.08 (p = 0.42)
Task B	D = −0.11 hit = 44% (p = 0.06)	r = 0.15 (p = 0.13)
Task C	D = 0.01 hit = 50% (p = 0.53)	r = −0.01 (p = 0.94)
Task D	D = −0.09 hit = 45% (p = 0.08)	r = 0.13 (p = 0.19)
Task E	D = 0.34 hit = 68% (p < 0.01)	r = −0.41 (p < 0.01)
Task F	D = 0.26, hit = 64% (p < 0.01)	r = −0.35 (p<0.01)
Task G	D = 0.24, hit = 63%, (p < 0.01)	r = −0.29(p < 0.01)
Task H	D = 0.37, hit = 70%, (p < 0.01)	r = −0.50 (p < 0.01)
Task I	D = 0.22, hit = 62% (p < 0.01)	r = −0.29 (p < 0.01)
ClusterE-F	Hit=70%	r=0.45

Skill testing will probably remain the best (however costly) option to identify high software development skill.

The second best may, however, be to ask them to predict the effort of tasks with «hidden simplicity» (looking more complex than they are, especially for those with low skill)