Software Developers' Work Habits and Expertise

Sketching, Code Plagiarism, and Expertise Development

Sebastian Baltes

@s_baltes

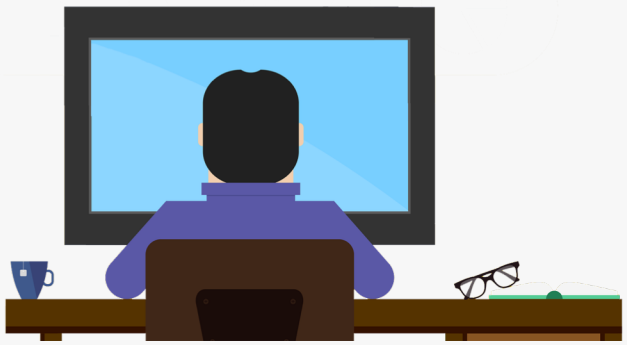
Universität Trier

empirical-software.engineering

Habit

„a **settled tendency** or **usual manner of behavior**“

https://www.merriam-webster.com/dictionary/habit



https://news.harvard.edu/gazette/story/2018/04/5-healthy-habits-may-increase-life-expectancy-by-decade-or-more/

Studied Habits

2013

Sketches and Diagrams in Practice

Sketching

SketchLink

How Developers Locate Performance Bugs

LivelySketches

Towards a Theory of Software Development Expertise

Expertise Development

Code Plagiarism

stackoverflow Code Snippets in GitHub Projects

2018

Studied Habits

2013

Sketches and Diagrams in Practice

Sketching

SketchLink

How Developers Locate Performance Bugs

LivelySketches

Towards a Theory of Software Development Expertise

Expertise Development

Code Plagiarism

stackoverflow Code Snippets in GitHub Projects

2018

Expertise Development

Towards a Theory of Software Development Expertise

Sebastian Baltes
University of Trier
Trier, Germany
research@sbaltes.com

Stephan Diehl
University of Trier
Trier, Germany
diehl@uni-trier.de

ESEC/FSE 2018

ABSTRACT

Software development includes diverse tasks such as implementing new features, analyzing requirements, and fixing bugs. Being an expert in those tasks requires a certain set of skills, knowledge, and experience. Several studies investigated individual aspects of software development expertise, but what is missing is a comprehensive theory. We present a first conceptual theory of software development expertise that is grounded in data from a mixed-methods survey with 335 software developers and in literature on expertise and expert performance. Our theory currently focuses on programming, but already provides valuable insights for researchers, developers, and employers. The theory describes important properties of software development expertise and which factors foster or hinder its formation, including how developers' performance may decline over time. Moreover, our quantitative results show that developers' expertise self-assessments are context-dependent and that experience is not necessarily related to expertise.

expert performance [78]. Bergersen et al. proposed an instrument to measure programming skill [9], but their approach may suffer from learning effects because it is based on a fixed set of programming tasks. Furthermore, aside from programming, software development involves many other tasks such as requirements engineering, testing, and debugging [62, 96, 100], in which a software development expert is expected to be good at.

In the past, researchers investigated certain aspects of software development expertise (SDExp) such as the influence of programming experience [95], desired attributes of software engineers [63], or the time it takes for developers to become "fluent" in software projects [117]. However, there is currently no theory combining those individual aspects. Such a theory could help structuring existing knowledge about SDExp in a concise and precise way and hence facilitate its communication [44]. Despite many arguments in favor of developing and using theories [46, 56, 85, 109], theory-driven research is not very common in software engineering [97].

https://empirical-software.engineering/projects/expertise/

Software Development Expertise?

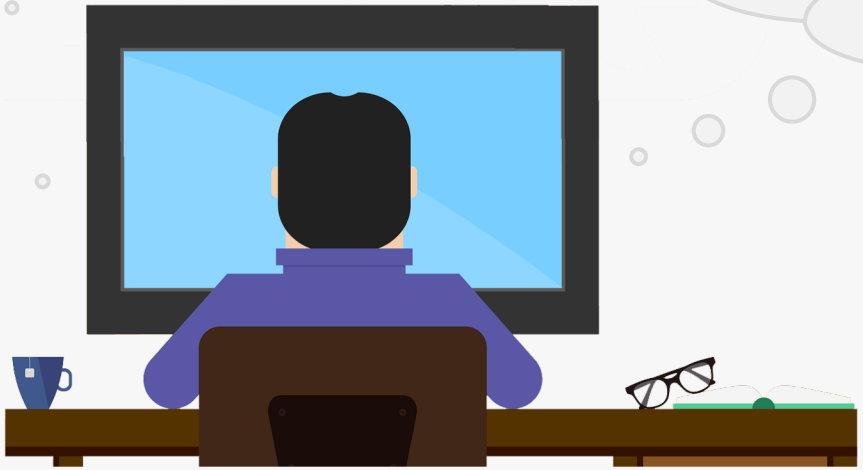
Implementing new features

Data structures

Testing

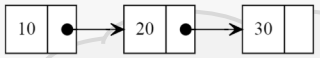
Communication

Debugging

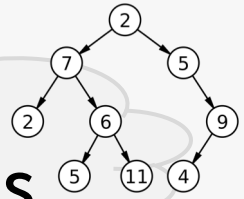


Software Development Expertise?

Implementing new features



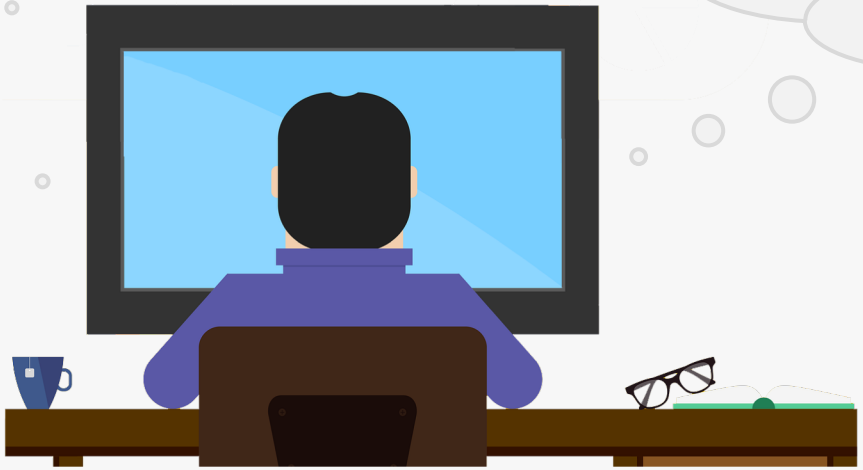
Data structures



Testing

Communication

Debugging




How to structure all those expertise-related aspects?


Which factors influence expertise development over time?

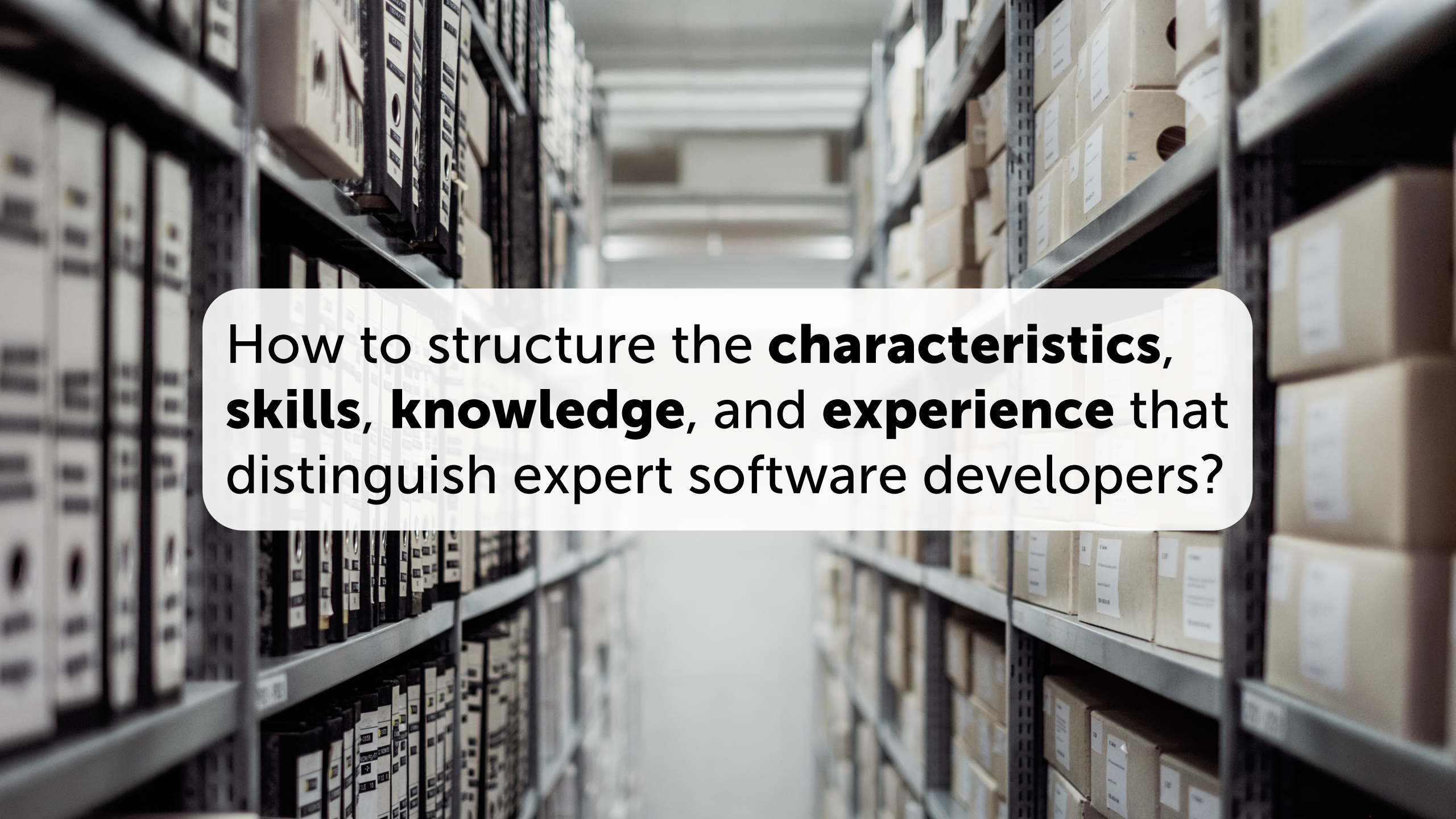
How are experience and expertise related?

Definitions

An expert is someone *"with the special **skill** or **knowledge** representing mastery of a **particular subject**"*

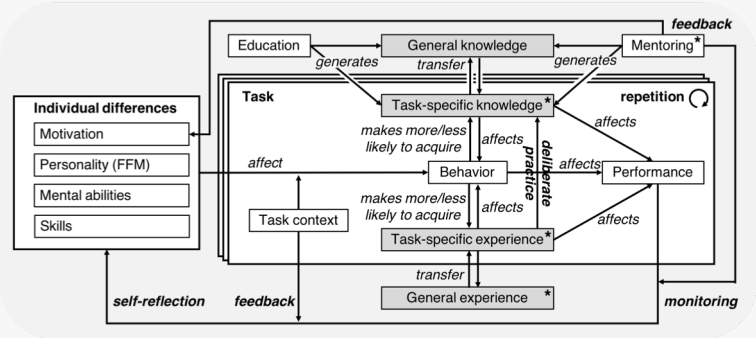
Expertise are *„the **characteristics**, **skills**, and **knowledge** that distinguish experts from novices and less **experienced** people."*

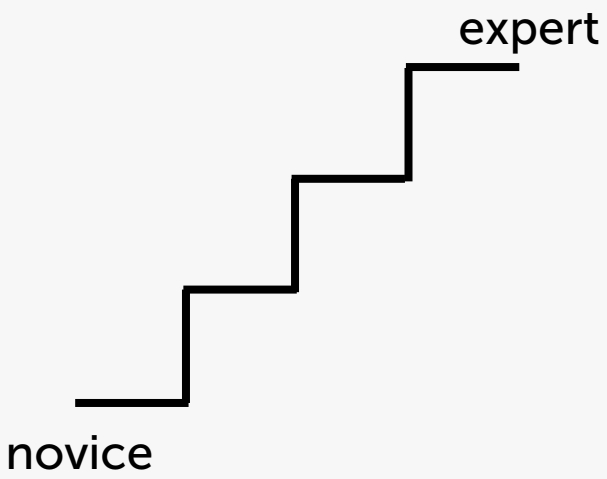
K. Anders Ericsson

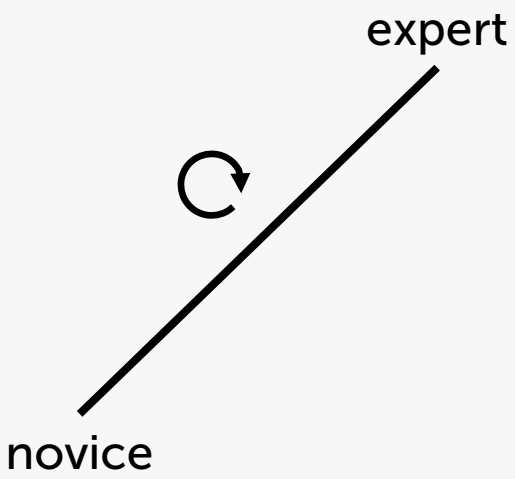

How to structure the **characteristics**, **skills**, **knowledge**, and **experience** that distinguish expert software developers?

Our Expertise Model

- **Task-specific** (e.g., writing code, debugging, testing)
- Focuses on **individual developers**
- **Process** view (repetition of tasks)
- Notion of **transferable knowledge and experience** from related fields or tasks
- **Continuum** instead of discrete expertise steps



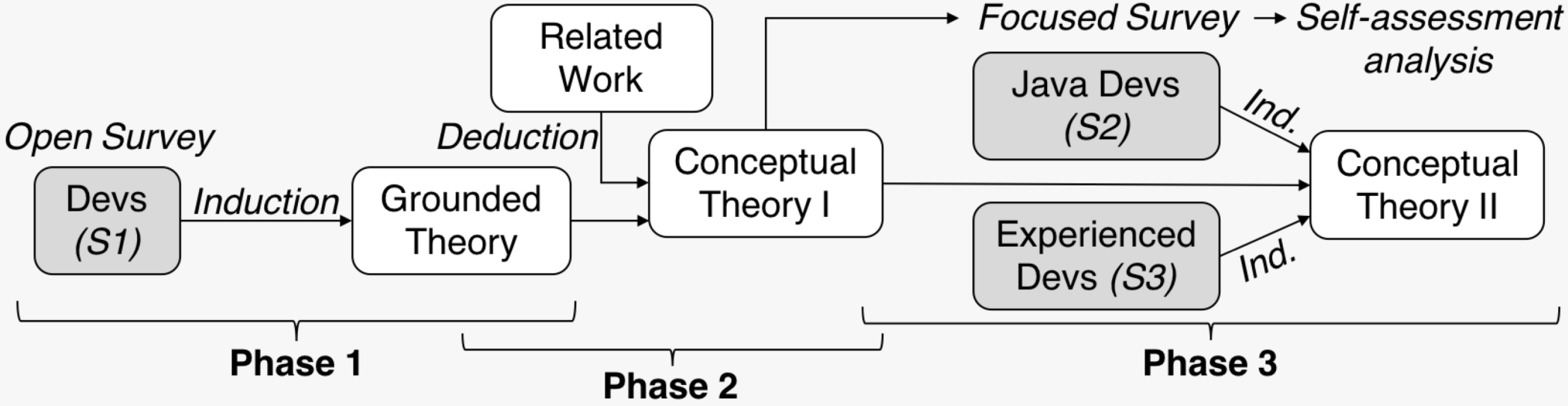




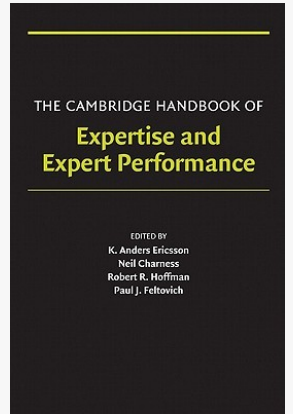
Theory Classification

- A **process theory** intends to explain and understand *"how an entity changes and develops"* over time (Ralph, 2018)
- In a **teleological process theory**, an entity *"constructs an envisioned end state, takes action to reach it, and monitors the progress"* (van de Ven and Poole, 1995)
- **Our theory:**
  - *Entity:*
    Individual software developer working on different software development tasks
  - *Envisioned end state:*
    Being an expert in (some of) those tasks

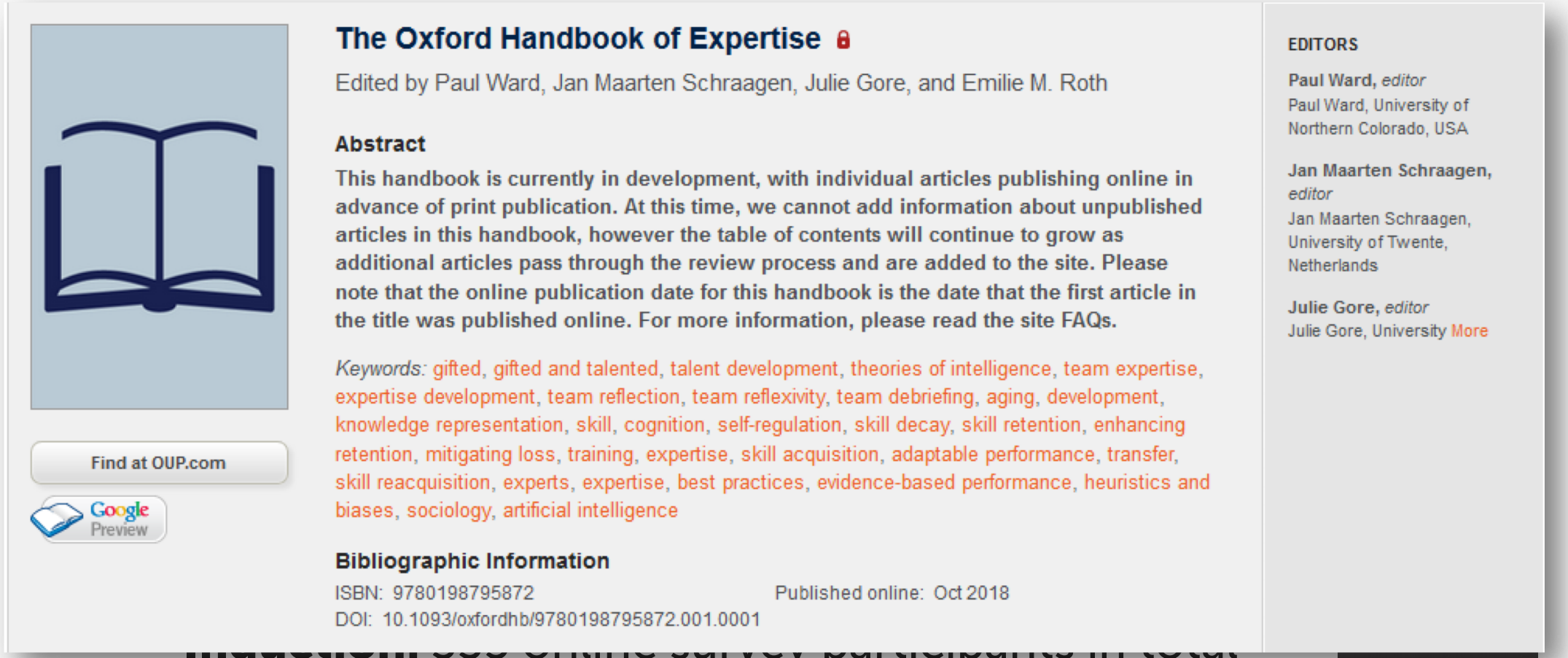
Research Design



- **Induction:** 335 online survey participants in total
- **Deduction:** Main source *"Cambridge Handbook of Expertise and Expert Performance"*



Research Design



The Oxford Handbook of Expertise

Edited by Paul Ward, Jan Maarten Schraagen, Julie Gore, and Emilie M. Roth

Abstract

This handbook is currently in development, with individual articles publishing online in advance of print publication. At this time, we cannot add information about unpublished articles in this handbook, however the table of contents will continue to grow as additional articles pass through the review process and are added to the site. Please note that the online publication date for this handbook is the date that the first article in the title was published online. For more information, please read the site FAQs.

Keywords: gifted, gifted and talented, talent development, theories of intelligence, team expertise, expertise development, team reflection, team reflexivity, team debriefing, aging, development, knowledge representation, skill, cognition, self-regulation, skill decay, skill retention, enhancing retention, mitigating loss, training, expertise, skill acquisition, adaptable performance, transfer, skill reacquisition, experts, expertise, best practices, evidence-based performance, heuristics and biases, sociology, artificial intelligence

Bibliographic Information

ISBN: 9780198795872 Published online: Oct 2018

DOI: 10.1093/oxfordhb/9780198795872.001.0001

Find at OUP.com

Google Preview

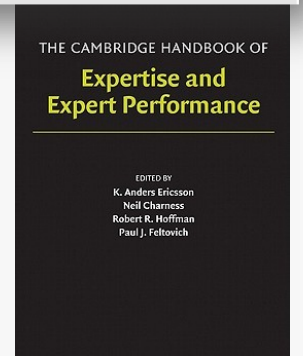
EDITORS

Paul Ward, *editor*
Paul Ward, University of Northern Colorado, USA

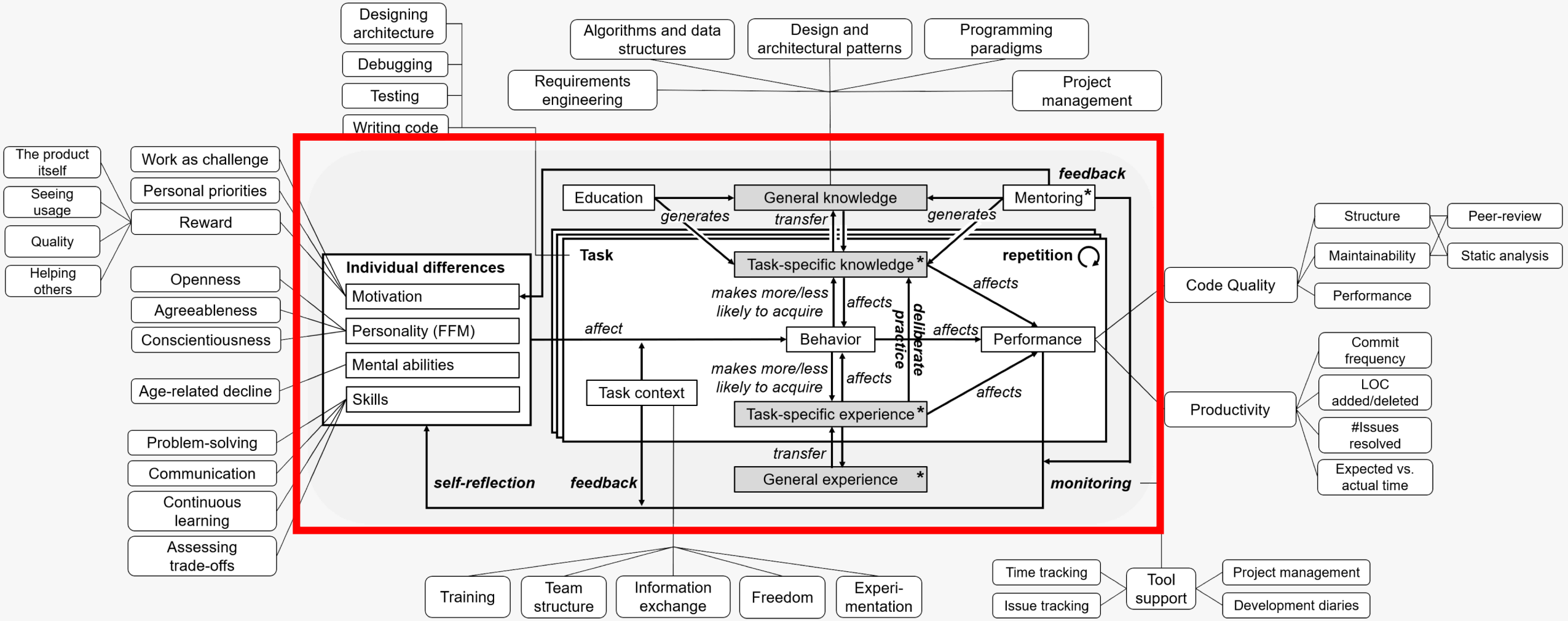
Jan Maarten Schraagen, *editor*
Jan Maarten Schraagen, University of Twente, Netherlands

Julie Gore, *editor*
Julie Gore, University More

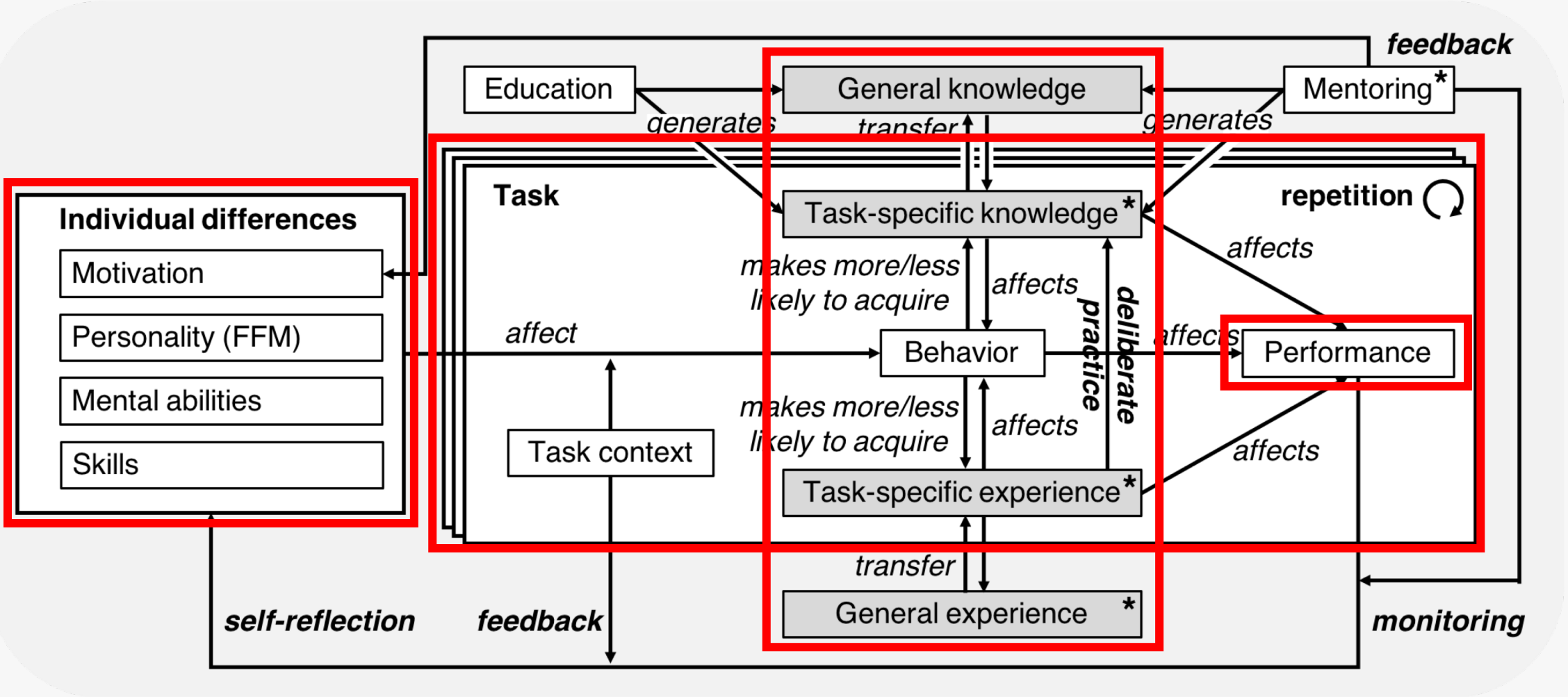
- **Induction:** 365 online survey participants in total
- **Deduction:** Main source *"Cambridge Handbook of Expertise and Expert Performance"*



Final Conceptual Theory



Final Conceptual Theory



Knowledge

- **Knowledge** is a *"permanent structure of information stored in memory"* (Robillard, 1995)
- Developer's knowledge base considered (most) important factor influencing **performance** (Curtis, 1984)
- Studies suggest that this knowledge base is *"highly **language dependent**",* but experts also have *"abstract, **transferable** knowledge and skills"* (Sonnentag et al., 2006)
- *"Semantic"* vs. *"syntactical"* knowledge (Shneiderman and Mayer, 1978)

Knowledge

- **Knowledge** is a *"permanent structure of information stored in memory"* (Robillard, 1995)
- Developer's knowledge base considered (most) important factor influencing **performance** (Curtis, 1984)
- Studies suggest t
  dependent", but
  knowledge and s
- *"Semantic"* vs. *"sy*

FIFTEEN YEARS OF PSYCHOLOGY IN SOFTWARE ENGINEERING:
INDIVIDUAL DIFFERENCES AND COGNITIVE SCIENCE

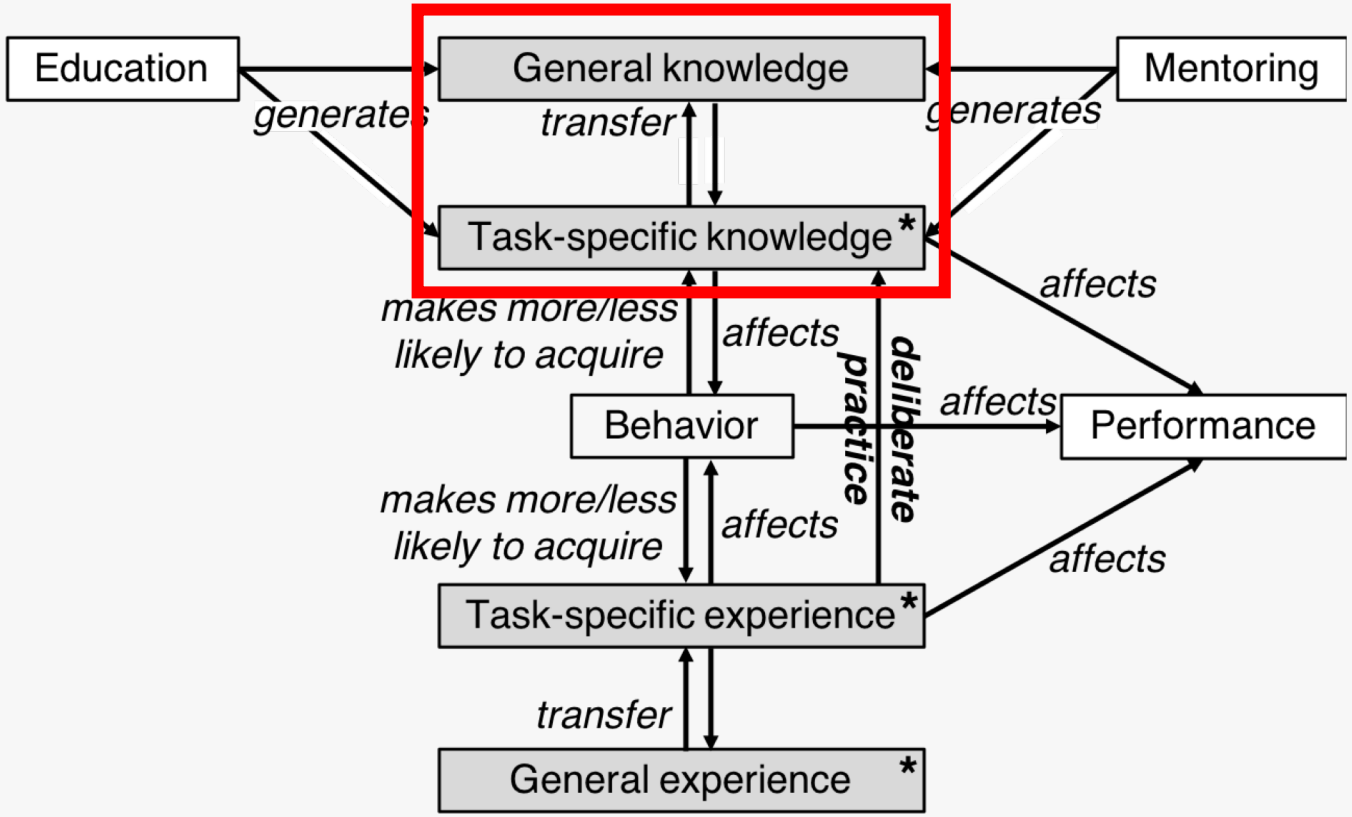
BILL CURTIS

ICSE 1984
(Orlando, FL, USA)

Microelectronics and Computer Technology Corporation (MCC)
Austin, Texas

Knowledge

Knowledge about *"paradigms [...], data structures, algorithms, computational complexity, and design patterns"*

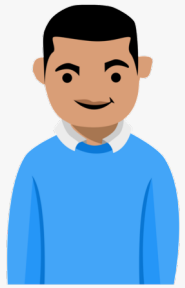


An *"intimate knowledge of the design and philosophy of the language"*

Experience

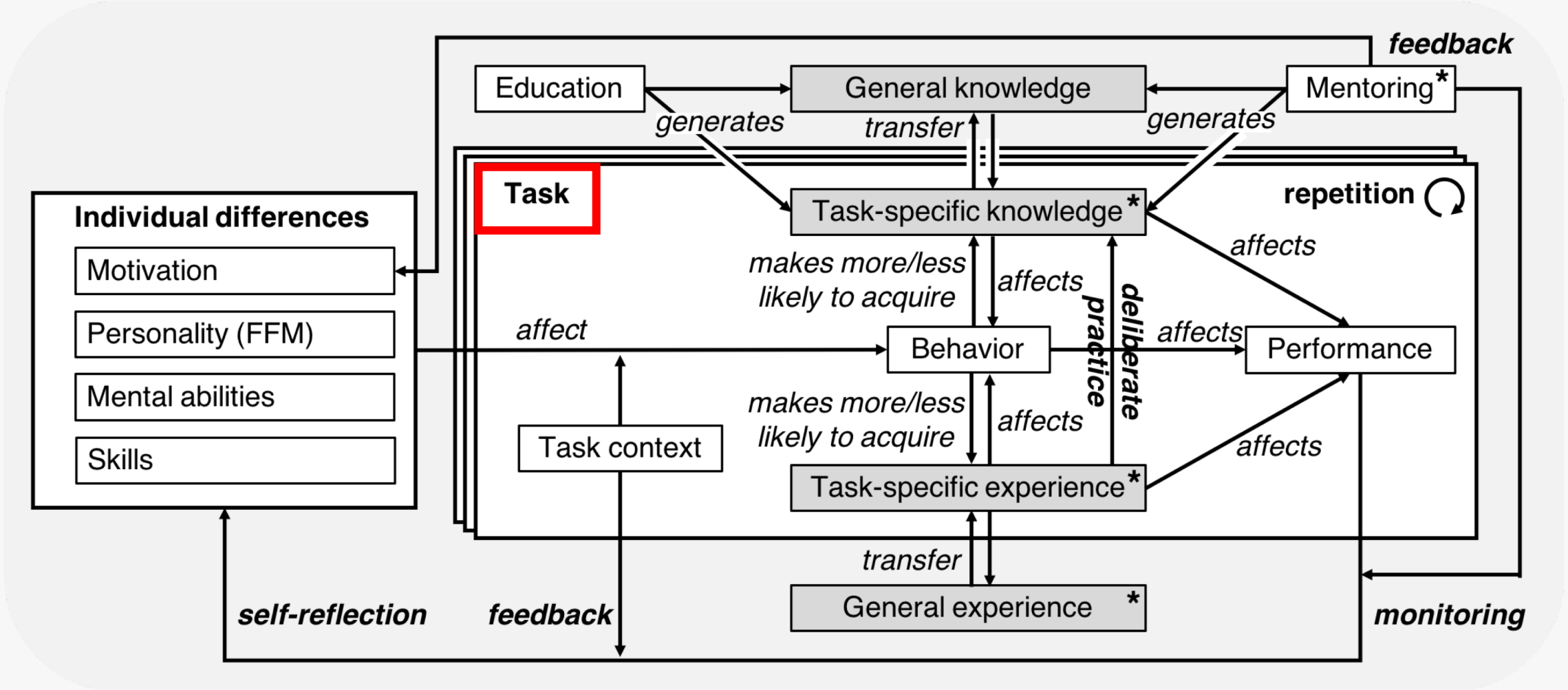
- Many participants mentioned not only the **quantity**, but also the **quality of experience**

Having built *„everything from small projects to enterprise projects“*



Having shipped *„a significant amount of code to production or to a customer“*

Final Conceptual Theory



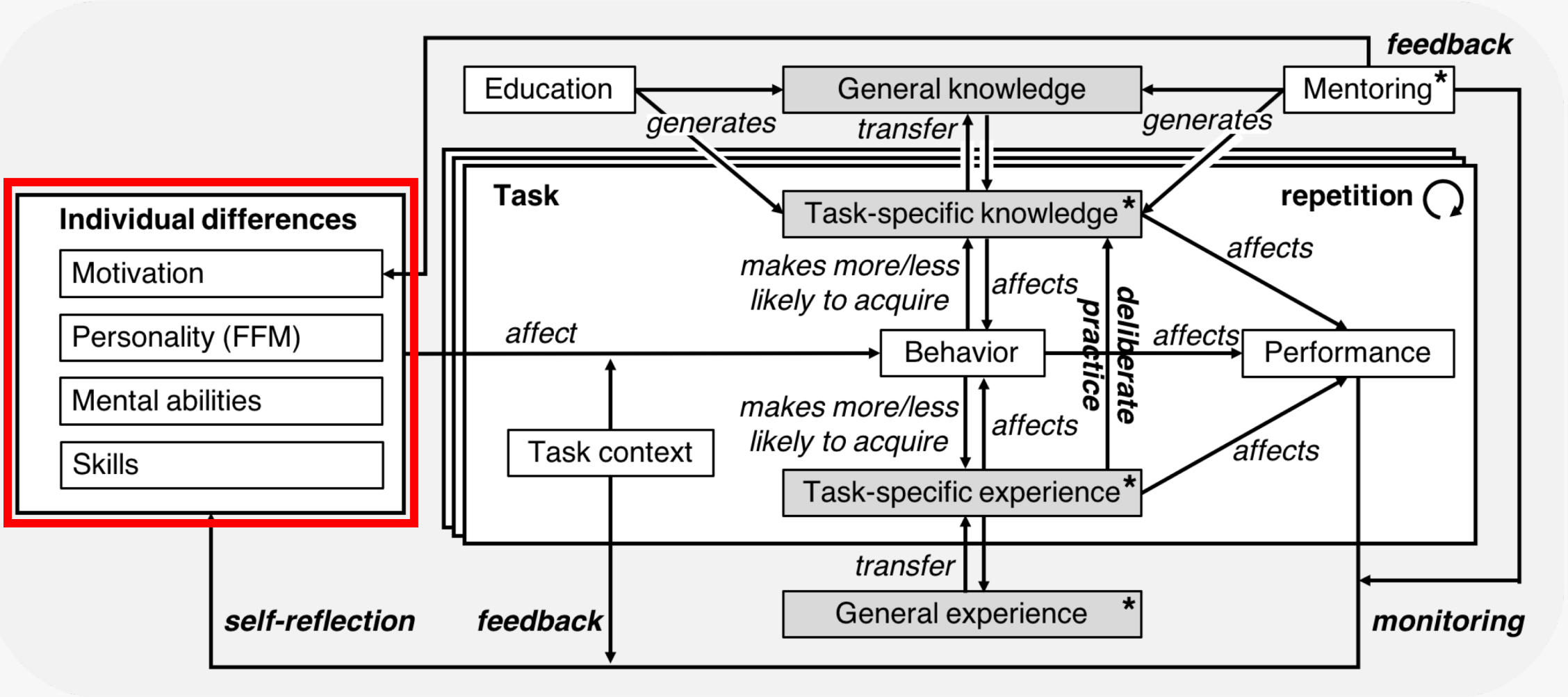
Tasks

- Asked participants to name the **three most important tasks** that a software development expert should be good at
- Most frequently mentioned:
  1. Designing a software architecture
  2. Writing source code
  3. Analyzing and understanding requirements
- Other mentioned tasks: testing, communicating, debugging

"Architecting the software in a way that allows flexibility in project requirements and future applications of the components"


Which factors influence expertise development over time?

Final Conceptual Theory

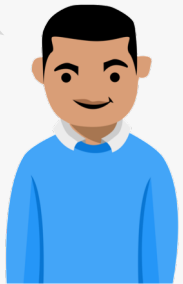


Individual Differences: Motivation

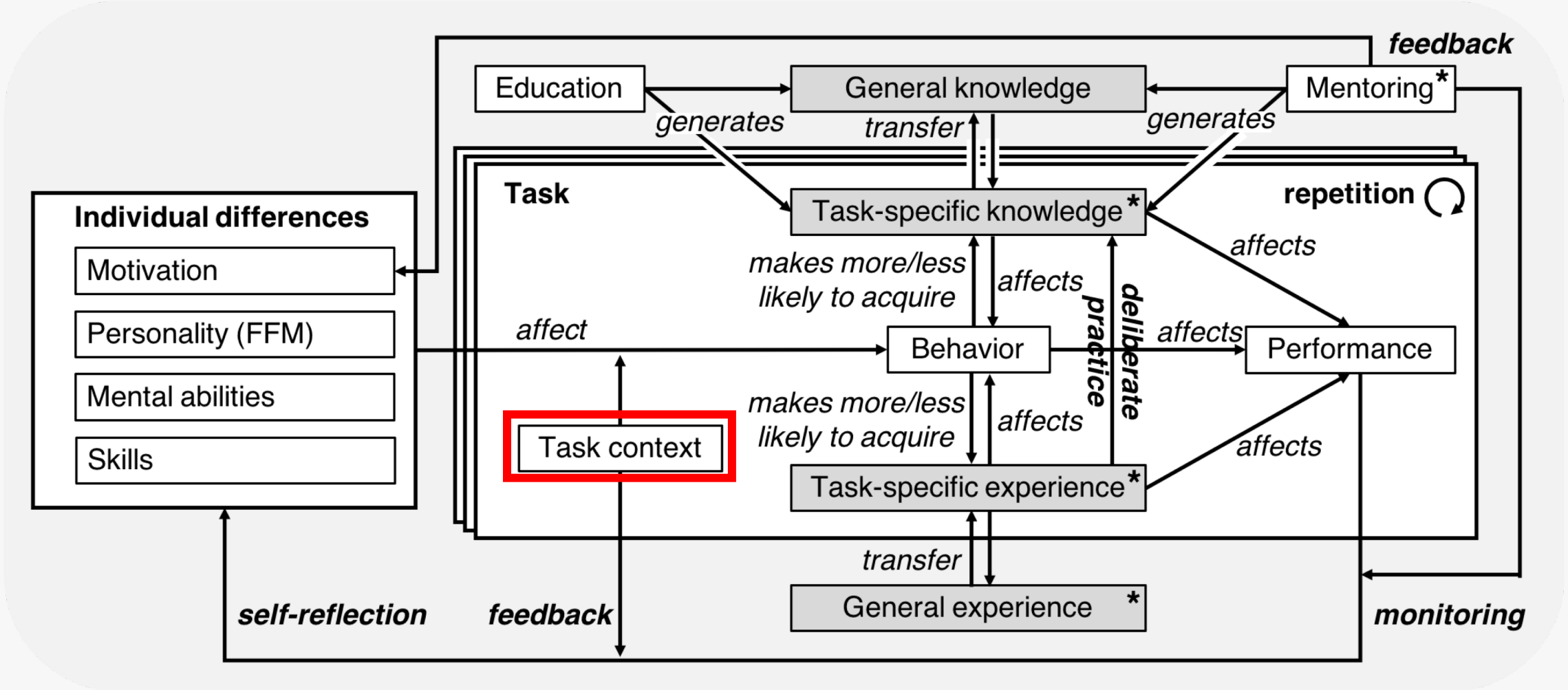
- Related work describes how **individual differences** affect expertise development
- Mental abilities and personality are relatively stable
- **Motivation can change** over time

- Many participants **intrinsically motivated:**
  - Problem solving
  - Seeing a high-quality solution
  - Creating something new
  - Helping others

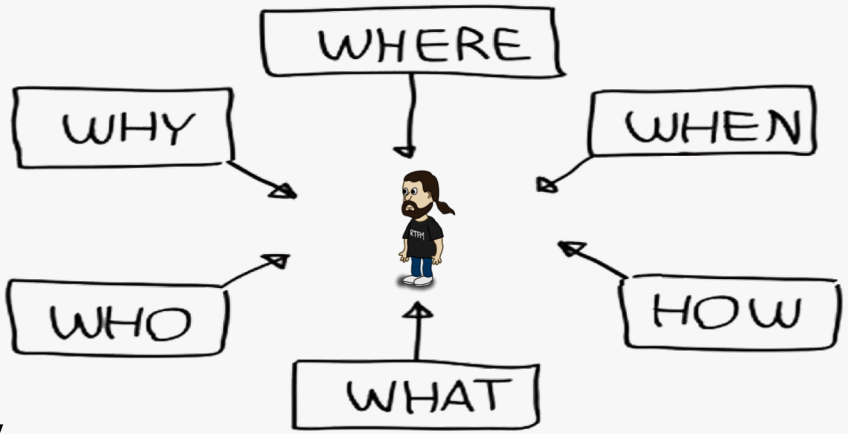
*"The initial design is fun, but what really is more rewarding is **refactoring**."*



Final Conceptual Theory



Task Context

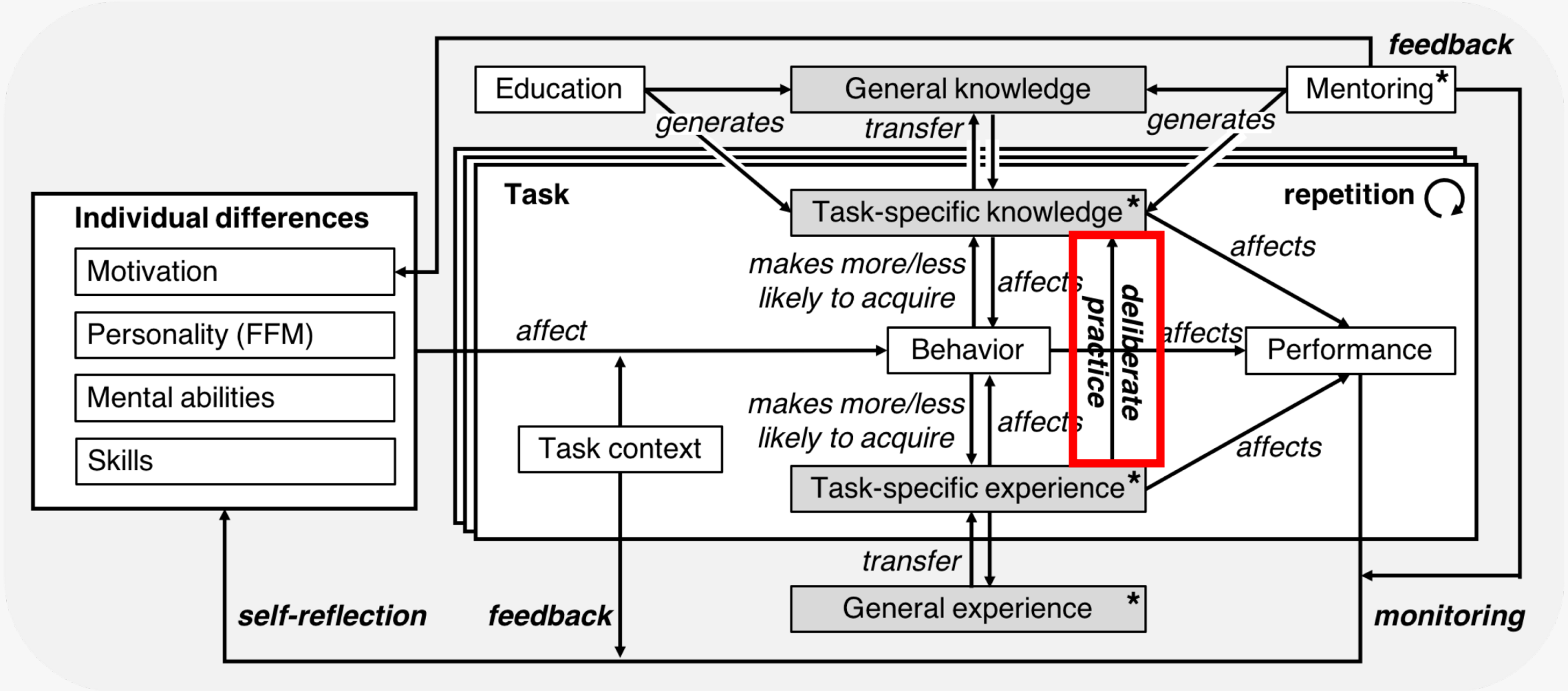


- Work **environment**
  (office, coworkers, customers etc.)
- Project **constraints**
  (external dependencies, time, etc.)
- Can either **foster or hinder** expertise dev.
- We asked: *What can employers do?*

1. Encourage learning
   (training courses, library, monetary incentives)
2. Encourage experimentation
   (side projects, being open to new ideas/technologies)
3. Improve information exchange
   (facilitate meetings, rotating between teams/projects)
4. Grant freedom
   (less time pressure)

http://kratinmobile.com/blog/wp-content/uploads/2014/09/context3.png

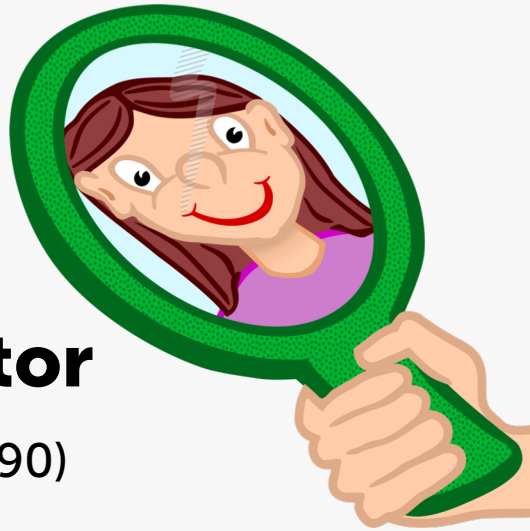
Final Conceptual Theory



Deliberate Practice

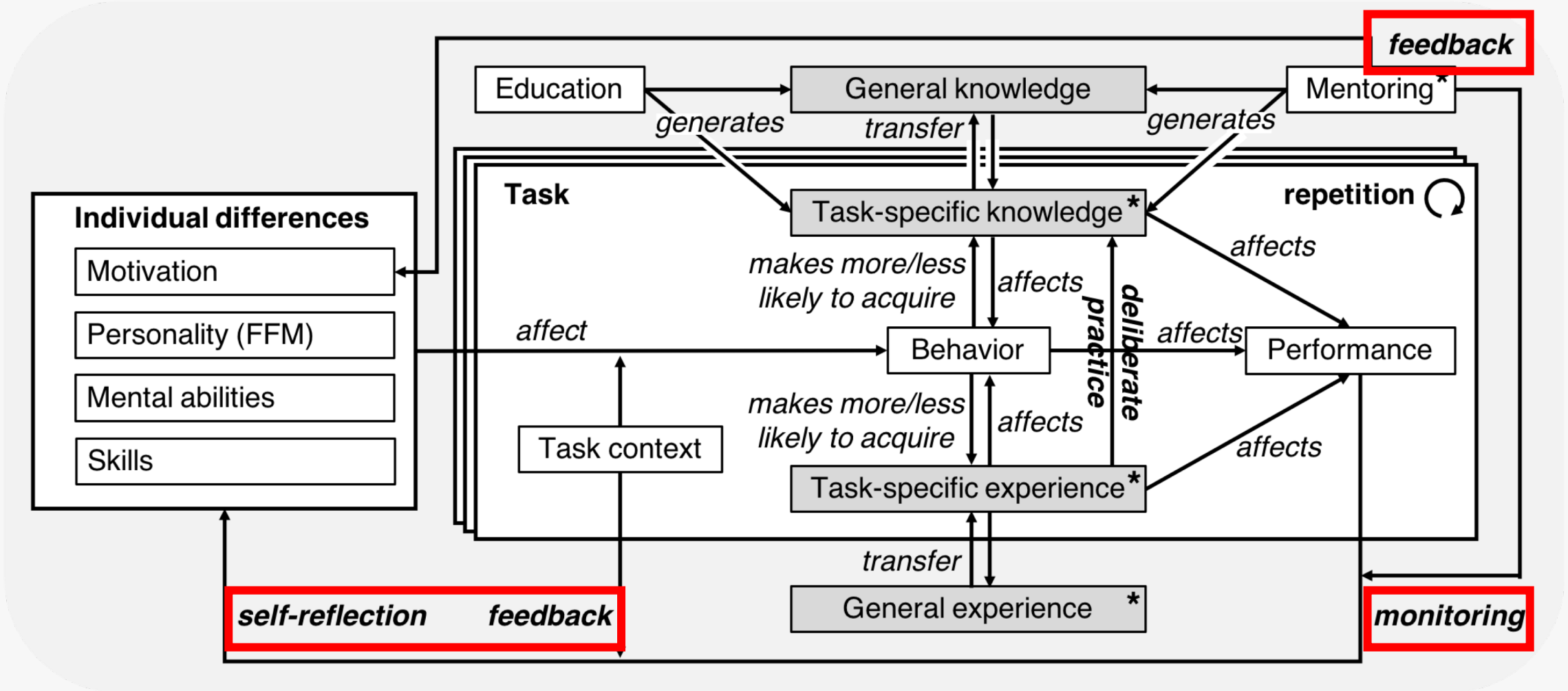
- Having **more experience** does not automatically lead to **better performance** (Ericsson et al., 1993)
- Performance may even **decrease** over time (Feltovich, 2006)
- Length of experience only weak correlate of job performance (Ericsson, 2006)
- Deliberate practice: *„**Prolonged efforts to improve** performance while negotiating motivational and external constraints“* (Ericsson et al., 1993)

Deliberate Practice: Self-Reflection

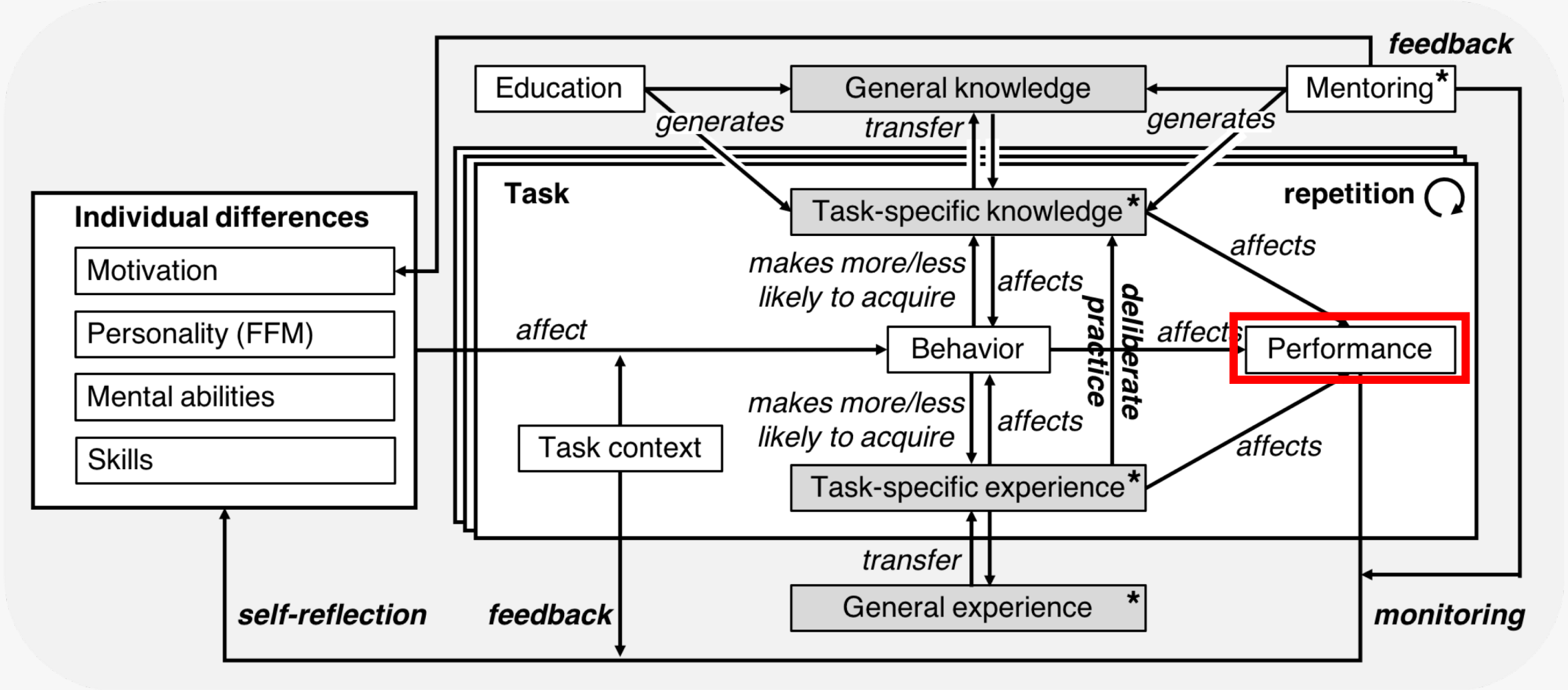


- **(Self-)reflection** and **feedback** important to **monitor** progress towards goal achievement (Locke and Latham, 1990)
- *"[T]he more **channels of accurate and helpful feedback** we have access to, the better we are likely to perform."* (Tourish and Hargie, 2003)
- **38.7%** of our participants reported that they **regularly monitor** their software development activity
- **Mentors**, teachers, and peers are an important sources for feedback

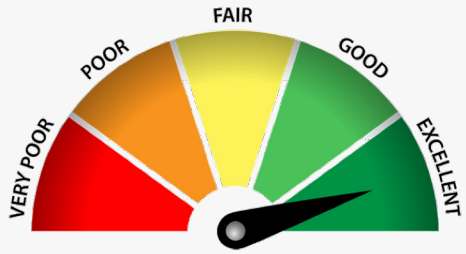
Final Conceptual Theory



Final Conceptual Theory



Performance



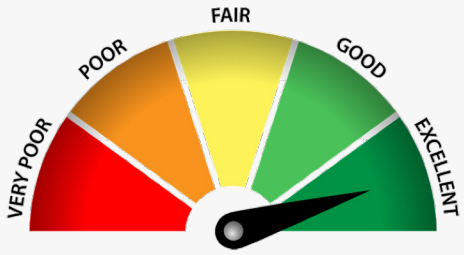
Scope of this work:

- We do **not** treat performance as a **dependent variable** that we try to explain or predict for individual tasks
- We consider different **performance monitoring** approaches to be a means for feedback and self-reflection

Long-term goal:

- Build **variance theory** for explaining and predicting the development of expertise

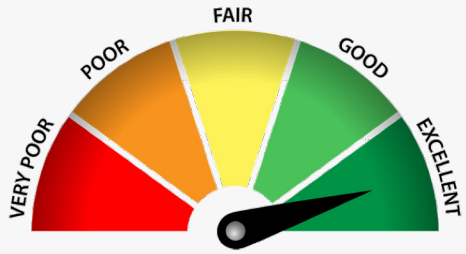
Performance



- Participants described different **properties of expert's source code** (well-structured, readable, maintainable, etc.)

„Everyone can write [...] code which a machine can read and process but the key lies in writing concise and understandable code which [...] ***people who have never used that piece of code before [can read]****."*

Expert Performance

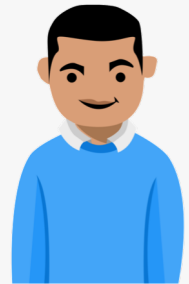


- In some areas (e.g., chess), there exist **representative tasks** and **objective criteria** for identifying experts
- Software development includes **many different tasks**
- Much more **difficult** to find objective measures for quantifying software development expert performance

Performance Decline

- Goal: Identify factors **hindering** expertise development
- **41.5%** of participants observed a **significant performance decline** over time (for themselves or others)
- Reasons:
  - Demotivation
  - Changes in the work environment
  - Age-related decline
  - Changes in attitude
  - Shifting towards other tasks

*"I perceived an **increasing procrastination** in me and in my colleagues, by **working on the same tasks** over a relatively long time [...] **without innovation and environment changes**."*



Age-Related Performance Decline

"For myself, it's mostly the effects of aging on the brain. At age 66, ***I can't hold as much information short-term memory****, for example. [...] I can compensate for a lot of that by writing simpler functions with clean interfaces. The results are still good, but* ***my productivity is much slower than when I was younger****."*

software architect, age 66

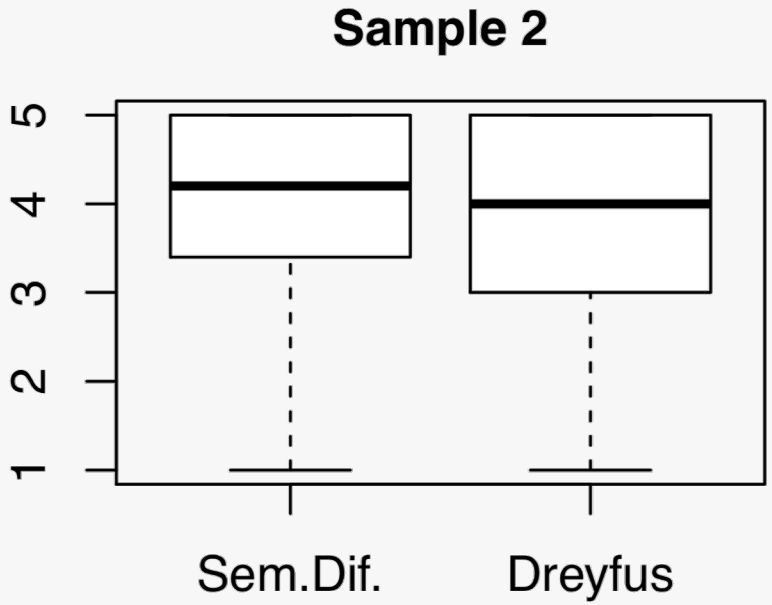
"Programming ability is based on **desire to achieve**. In the early years, it is a sort of **competition**. [...] I found that I lost a significant amount of my focus as I became 40, and started **using drugs such as ritalin** to enhance my abilities. This is pretty common among older programmers."

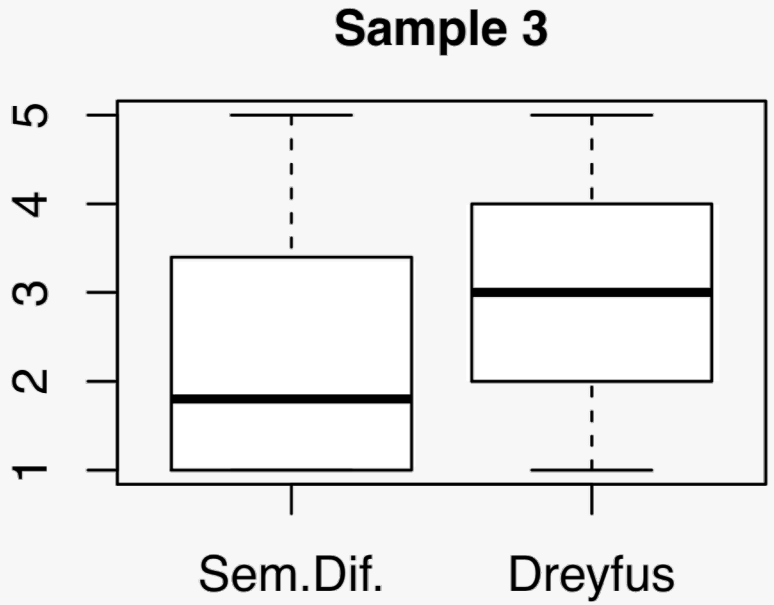
software developer, age 60

How are experience and expertise related?

Experience vs. Expertise

- Self-assessment with **semantic differential** (novice to expert) and **Dreyfus expertise model**
- More experienced developers **adjusted** their ratings when context was provided, less experienced not

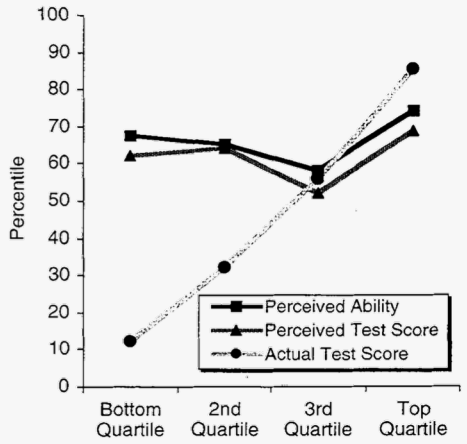




Experience vs. Expertise

- Analyzed correlation of experience (years) and self-assessed expertise and found **no consistent results**

- Possible explanation: **Dunning-Kruger effect**
  - Participants with a high skill-level underestimate their ability and performance relative to their peers
  - Context helped experienced developers to adjust their ratings to be more accurate



Experience vs. Expertise

- A
  a
- I

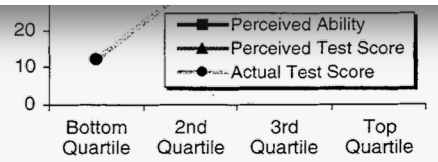
Journal of Personality and Social Psychology
1999, Vol. 77, No. 6, 1121–1134

0022-3514/99/$3.00

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

Justin Kruger and David Dunning
Cornell University

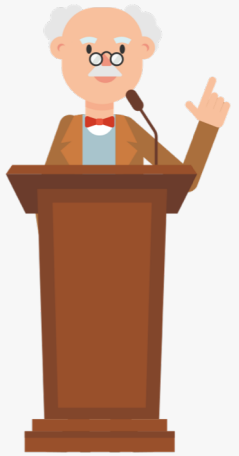
People tend to hold overly favorable views of their abilities in many social and intellectual domains. The authors suggest that this overestimation occurs, in part, because people who are unskilled in these domains suffer a dual burden: Not only do these people reach erroneous conclusions and make unfortunate choices, but their incompetence robs them of the metacognitive ability to realize it. Across 4 studies, the authors found that participants scoring in the bottom quartile on tests of humor, grammar, and logic grossly overestimated their test performance and ability. Although their test scores put them in the 12th percentile, they estimated themselves to be in the 62nd. Several analyses linked this miscalibration to deficits in metacognitive skill, or the capacity to distinguish accuracy from error. Paradoxically, improving the skills of participants, and thus increasing their metacognitive competence, helped them recognize the limitations of their abilities.



https://www.mirror.co.uk/news/weird-news/how-mcdonalds-takeaway-bag-ended-9664800

Summary for Researchers

- Can use our results when **designing studies** involving expertise **self-assessments** or our **theory building** approach
- Clear understanding what distinguishes novices and experts: **Provide** this **context** when asking for **self-assessed expertise** and later report it together with the results
- Can use theory to **design experiments** (first operationalizations described in paper)
- Future Work: Operationalization, develop **standardized description** of novice and expert for certain tasks

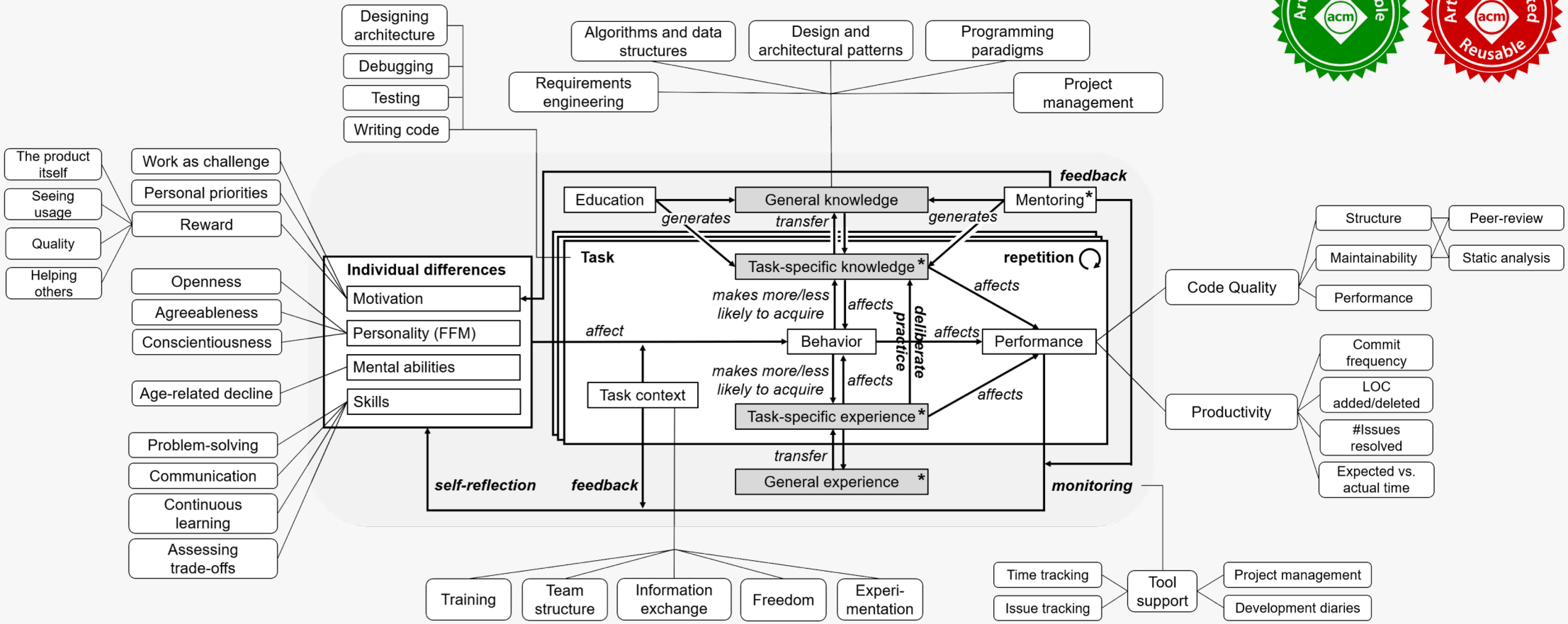


Summary for Developers

- See which **attributes** other developers assign to experts
- Learn which **behaviors** may lead to becoming a better software developer:
  - Deliberate practice
  - Have challenging goals
  - Build or maintain a supportive work environment (also for others)
  - Ask for feedback from peers
  - Reflect about what one knows and what not

Summary for Employers

- Learn what **(de)motivates** their employees:
  - Main motivation: problem solving
  - Main demotivation: non-challenging work
- Ideas on how to build supportive work environment **supporting self-improvement** of staff:
  - Good mix of continuity and change in software development process
  - Communicate clear visions, directions, and goals
  - Reward high-quality work wherever possible
  - Revisit information sharing in company
  - Facilitate meetings



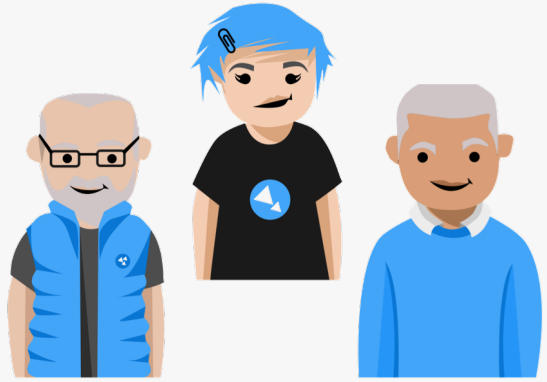
Sebastian Baltes

@s_baltes

expertise.sbaltes.com

Data and scripts available on Zenodo

Outlook

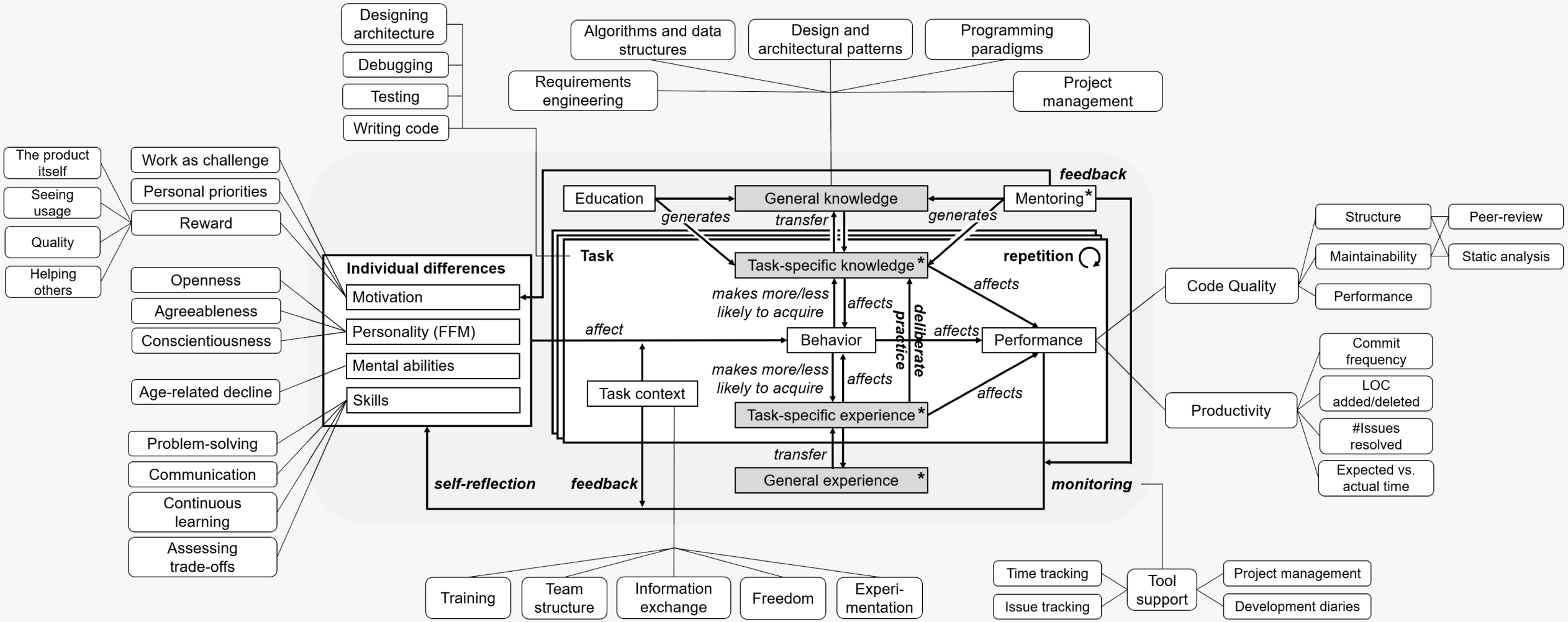


- Especially in industrialized countries, the **demographic change** leads to an older work force
- Study the **influence of aging** on software developers
  - Identify age-related **issues**
  - Develop **strategies** to address those issues
  - Prevent experienced developers from **dropping out** of software development
- Further research on the **factors fostering or hindering** expertise development
- Study expertise development from a **sociological perspective**

Questions

Ideas for hypothesis based on theory?

Possible operationalizations of concepts?



Research designs?

Tool support?