

Expertise in Software Engineering

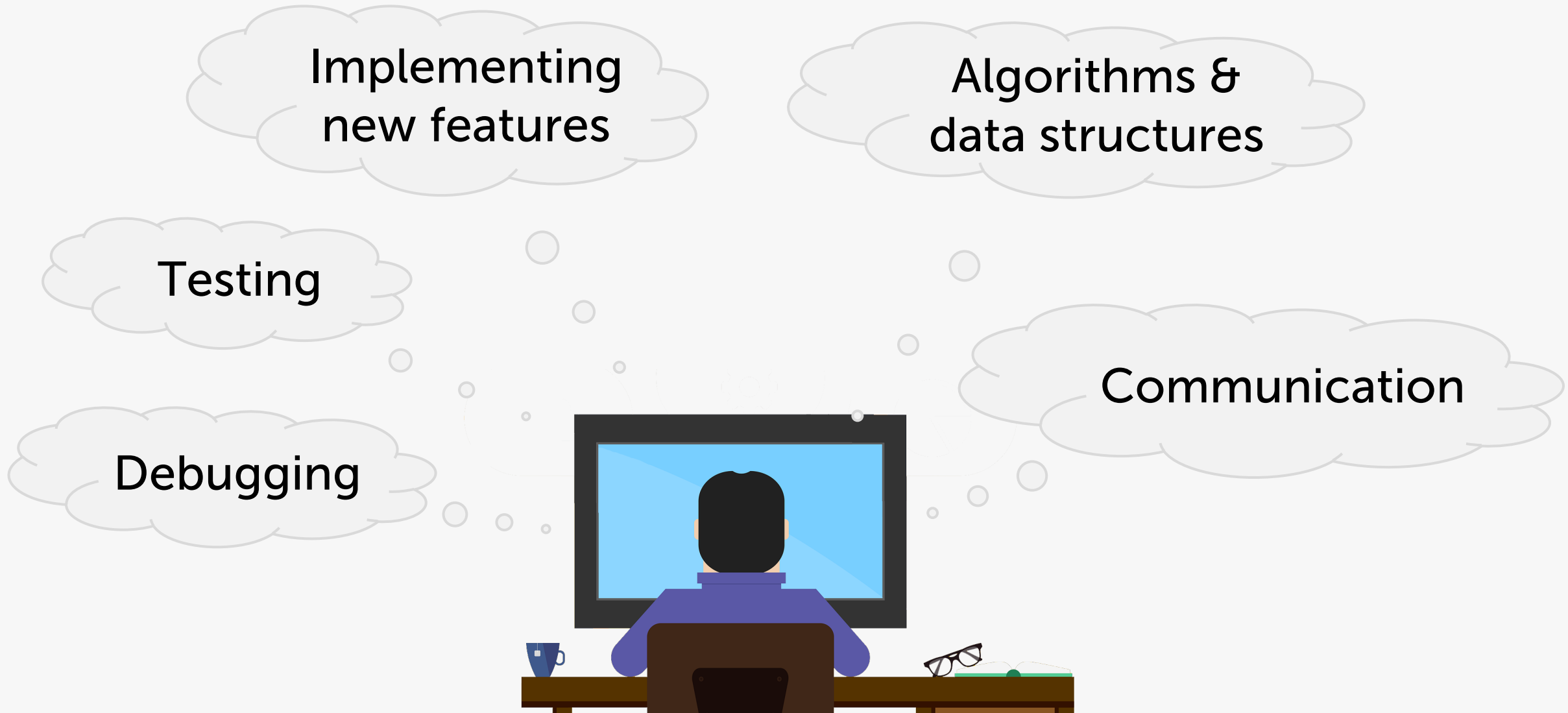
What can we learn from research in psychology?

Dr. Sebastian Baltes

 @s_baltes

 empirical-software.engineering

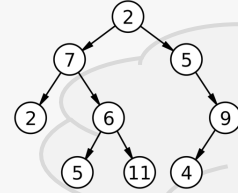
Software Development Expertise?



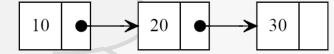
Software Development Expertise?



Implementing
new features



Algorithms &
Data structures



JUnit 5 Testing *jbehave*



Debugging



Communication





How to structure all those
expertise-related aspects?

Which factors influence expertise development over time?



How are experience and expertise related?



Definitions

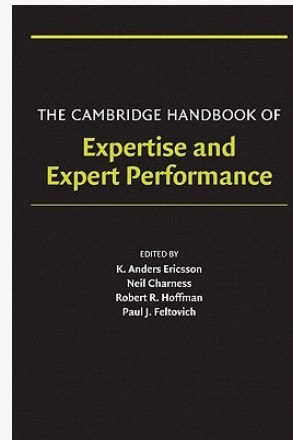
An **expert** is someone “with the special **skill** or **knowledge** representing mastery of a **particular subject**”



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

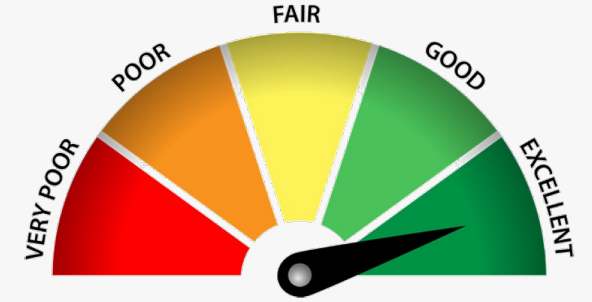


K. Anders Ericsson

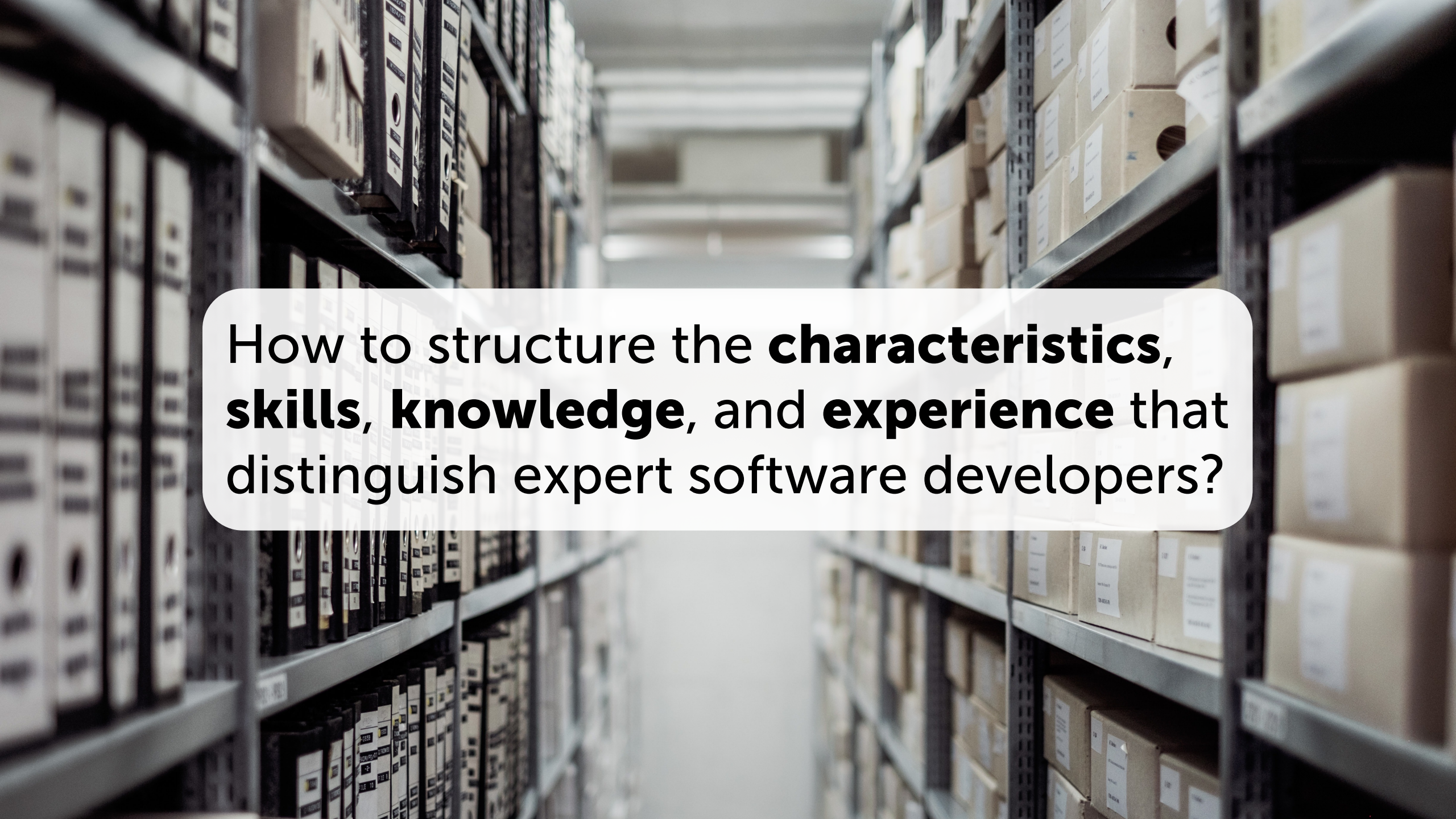


10,000 hours “rule” → not true
<https://www.goodlifeproject.com/podcast/anders-ericsson/>

Expert Performance



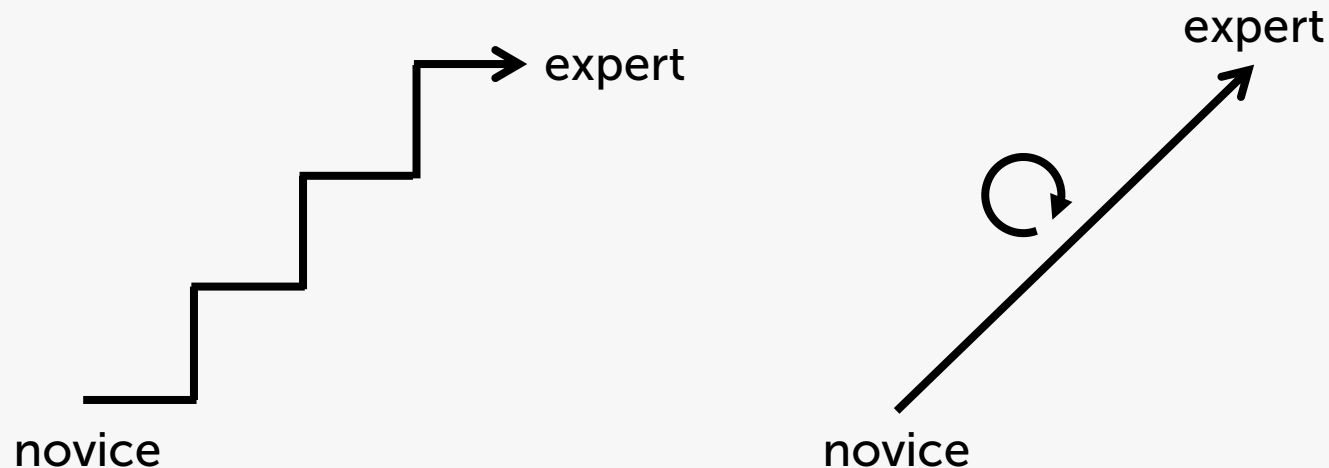
- In some areas (e.g., chess), there exist **representative tasks** and **objective criteria** for identifying experts
- Those are usually studied in (psychology) expertise research
- Software development includes **many different tasks**
- Much more **difficult** to find objective measures for quantifying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



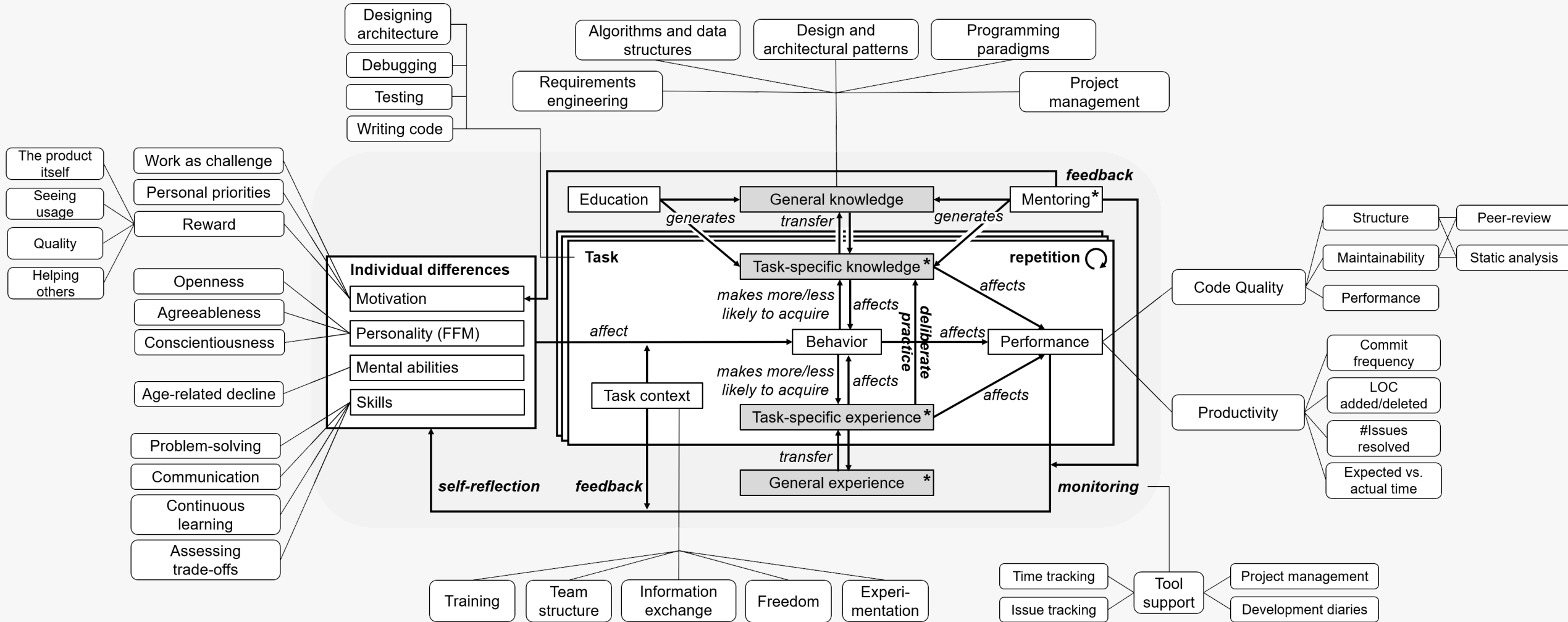
How to structure the **characteristics**, **skills**, **knowledge**, and **experience** that distinguish expert software developers?

Our Expertise Model

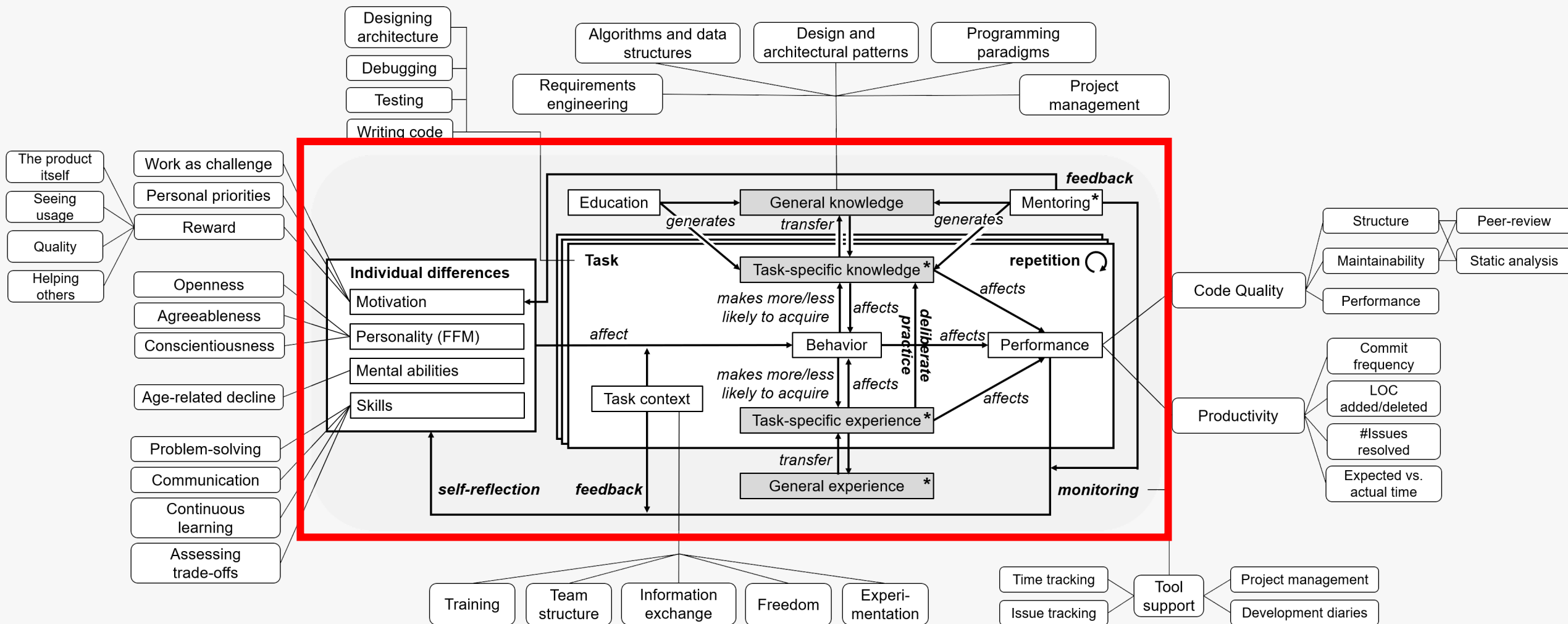
- **Task-specific** (e.g., writing code, debugging, testing)
- Focuses on **individual developers**
- **Process** view (repetition of tasks)
- Notion of **transferable knowledge and experience** from related fields or tasks
- **Continuum** instead of discrete expertise steps



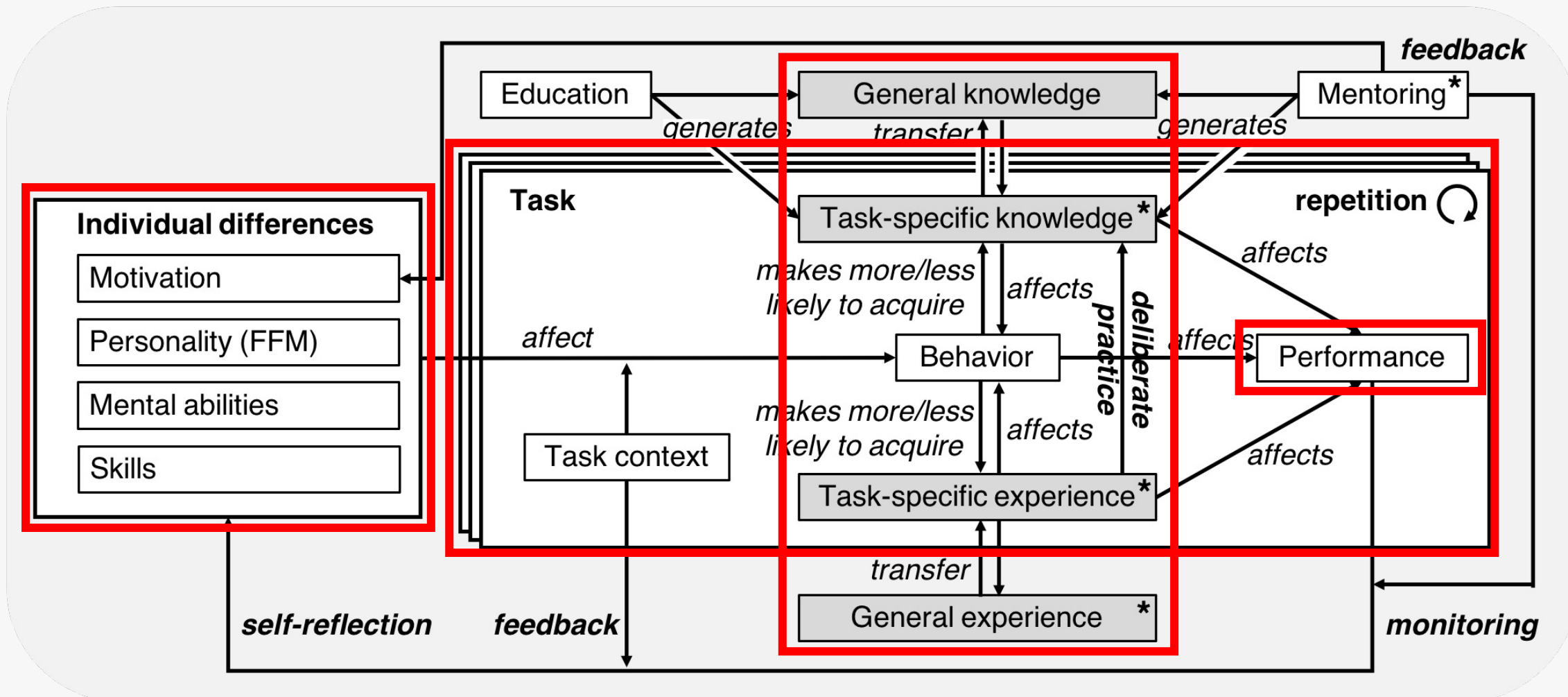
Final Conceptual Theory



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 that performance is **dependent** on knowledge and skills
- “*Semantic*” vs. “*syntactic*”

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

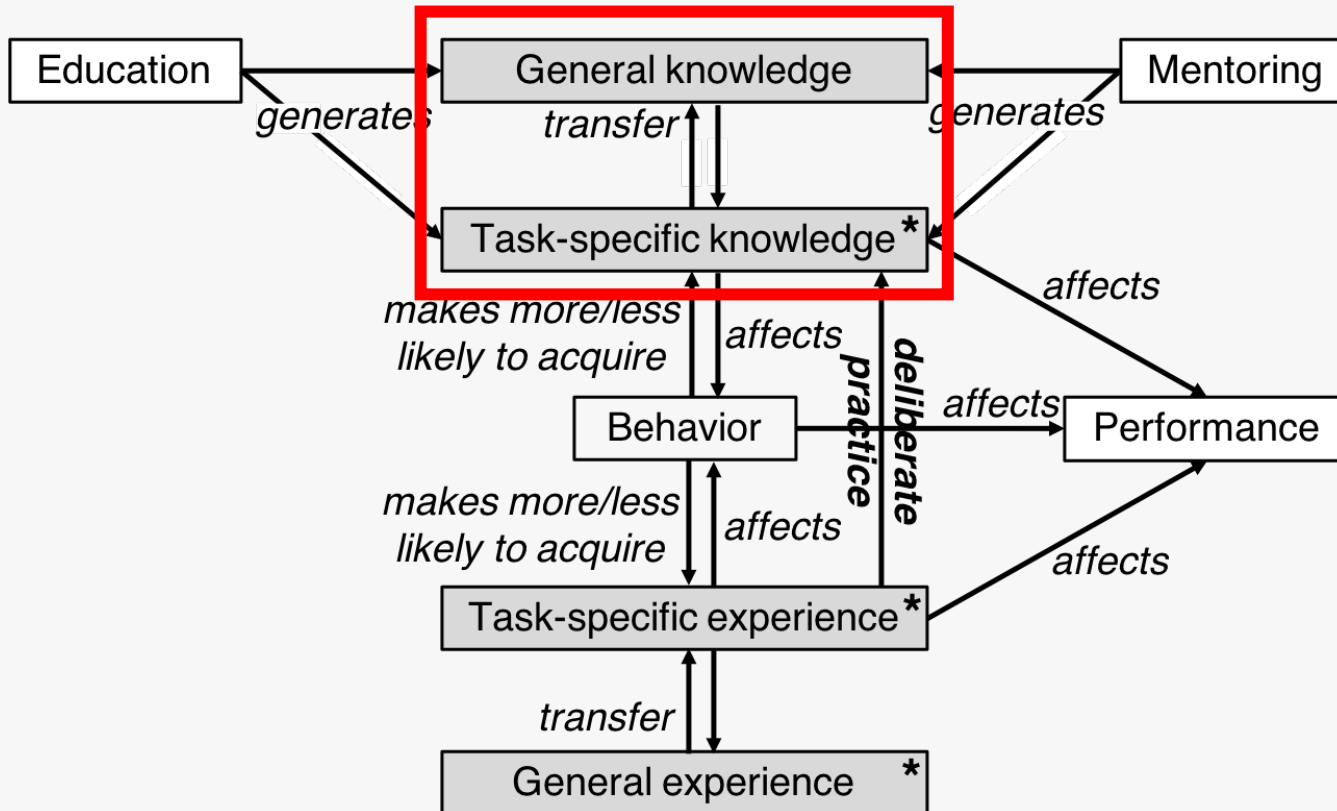
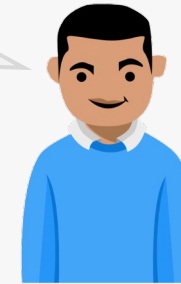
BILL CURTIS

ICSE 1984

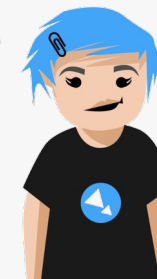
Microelectronics and Computer Technology Corporation (MCC)
Austin, Texas

Knowledge

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



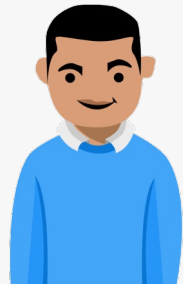
An “*intimate knowledge of the design and philosophy of the language*”



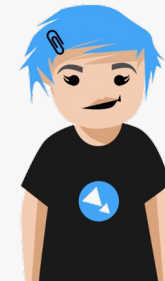
Experience

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

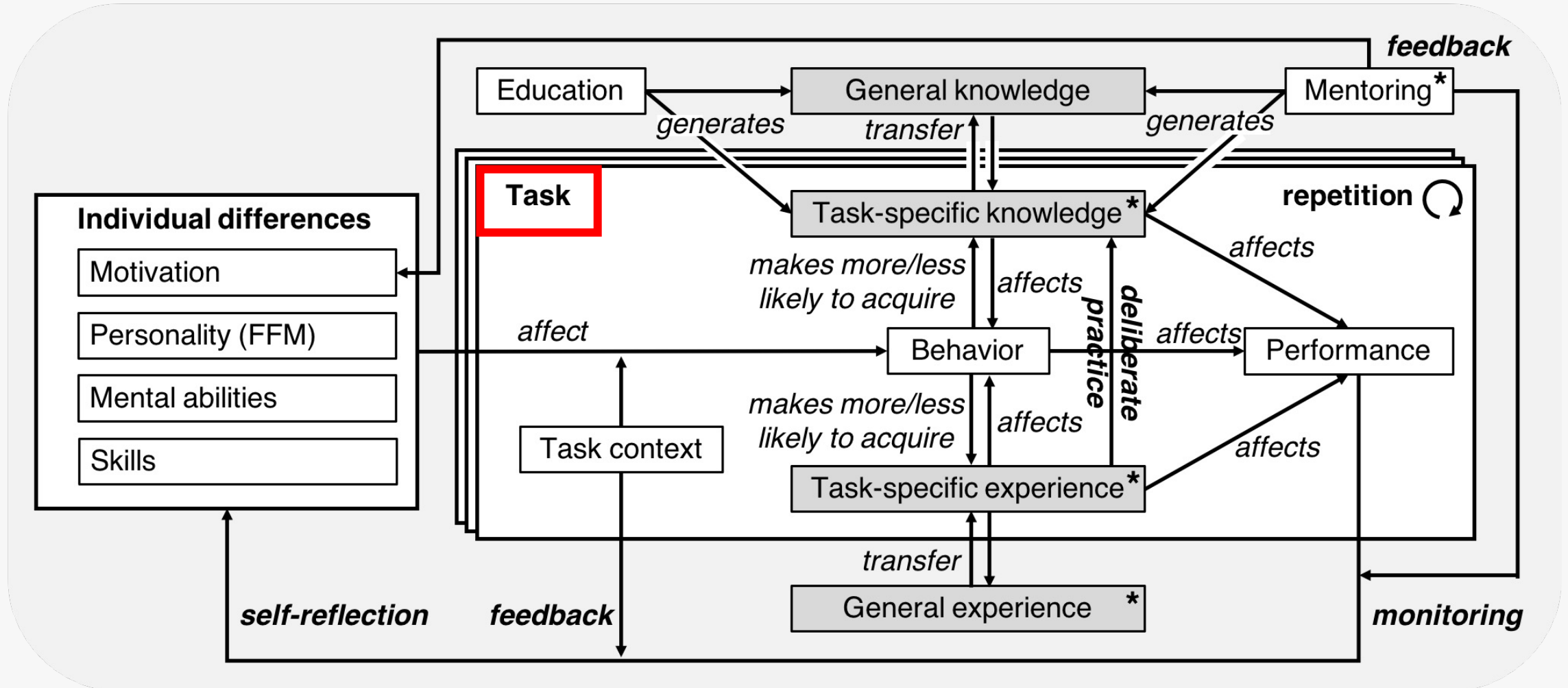
Having built „everything from small projects to enterprise projects“



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:
 - Designing a software architecture
 - Writing source code
 - Analyzing and understanding requirements
- Other mentioned tasks: testing, communicating, debugging

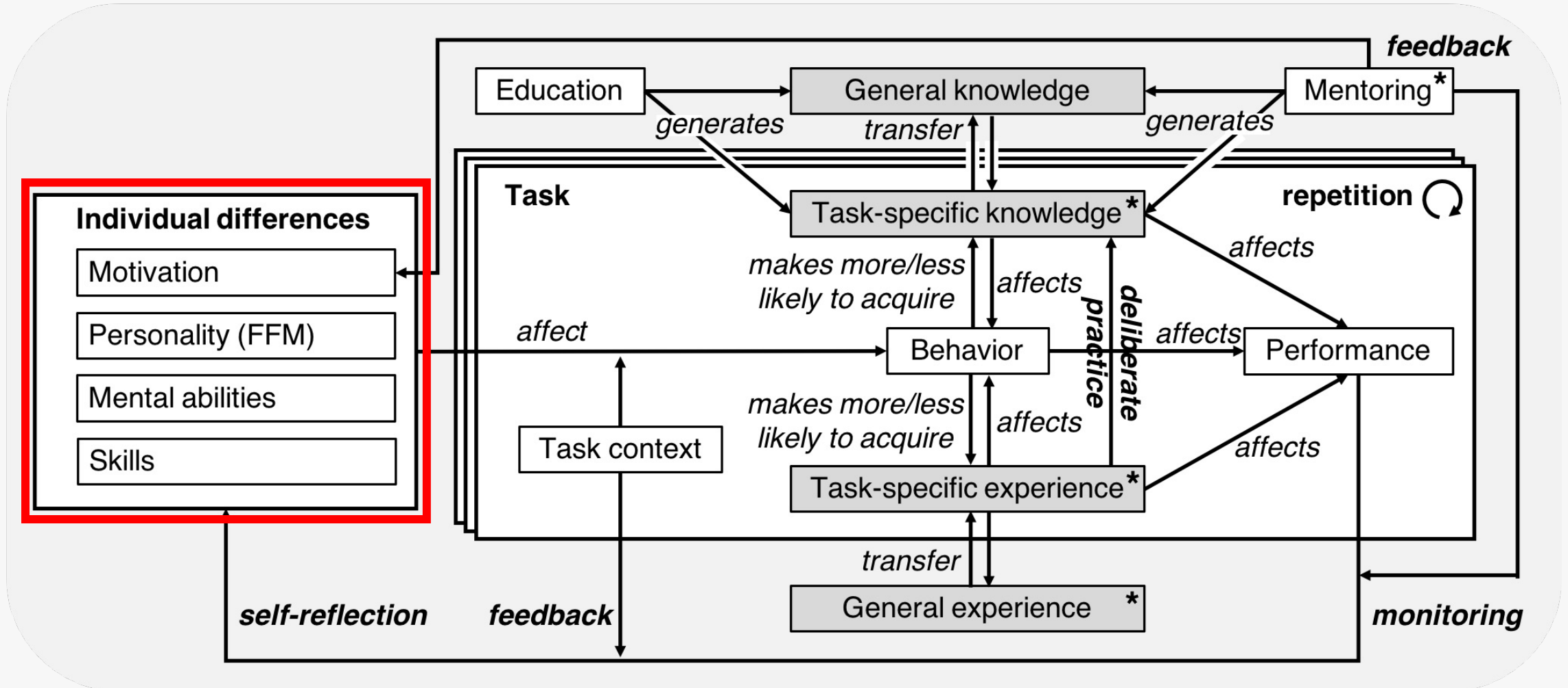
“Architecting the software in a way that allows flexibility in project requirements and future applications of the components”



Which factors influence expertise development over time?



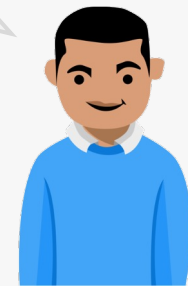
Final Conceptual Theory



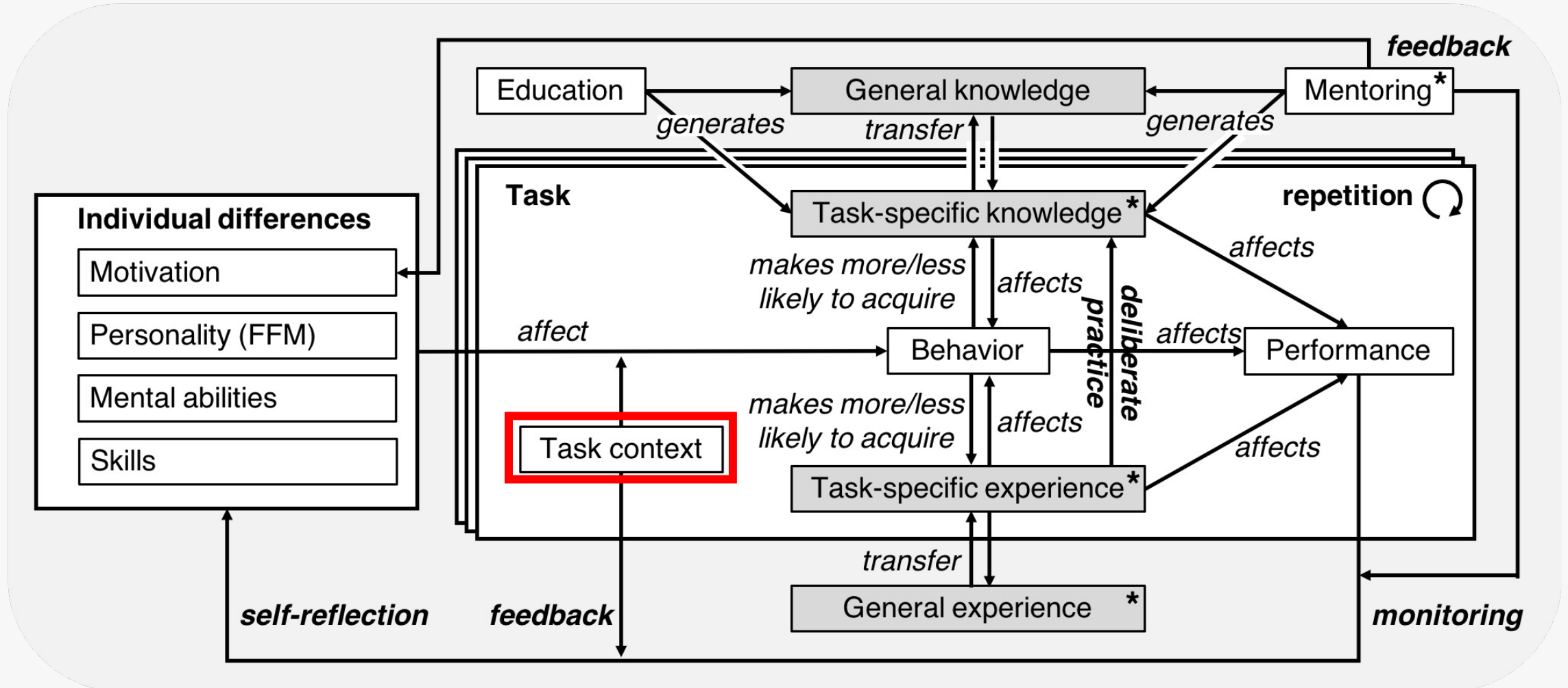
Individual Differences: Motivation

- 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**."*

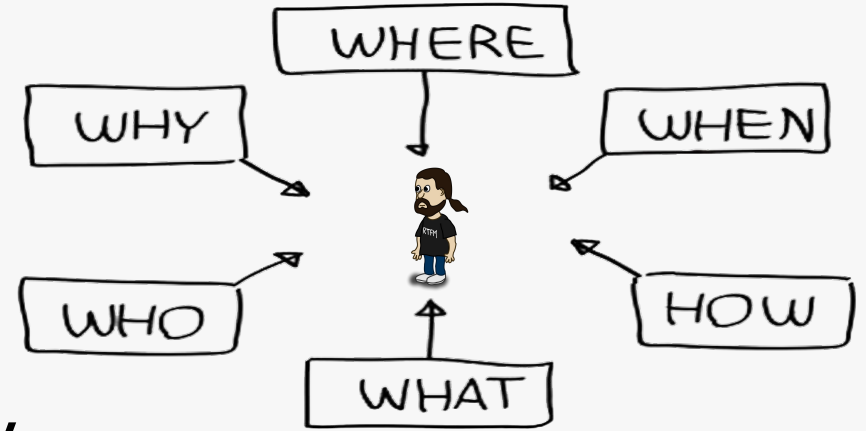


Final Conceptual Theory

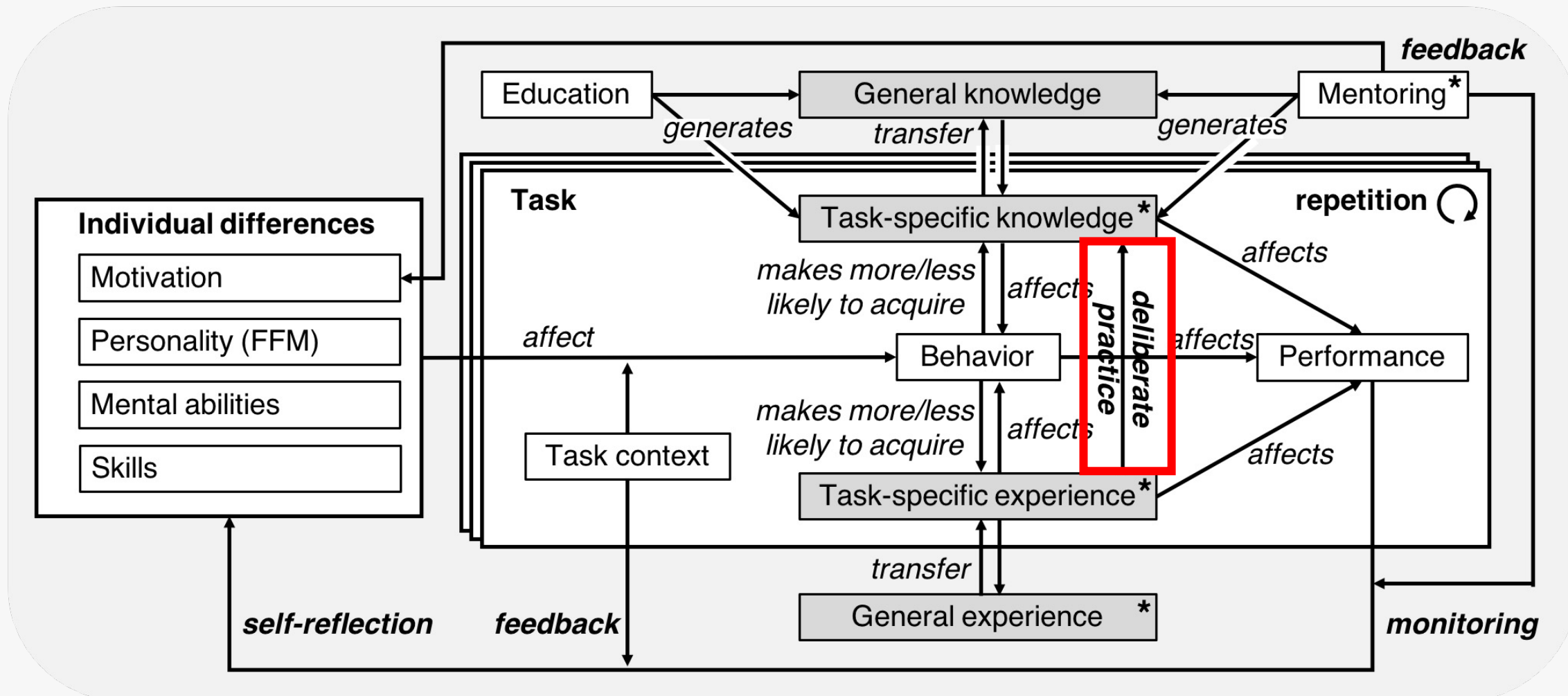


Task Context

- 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)



Final Conceptual Theory



Deliberate Practice



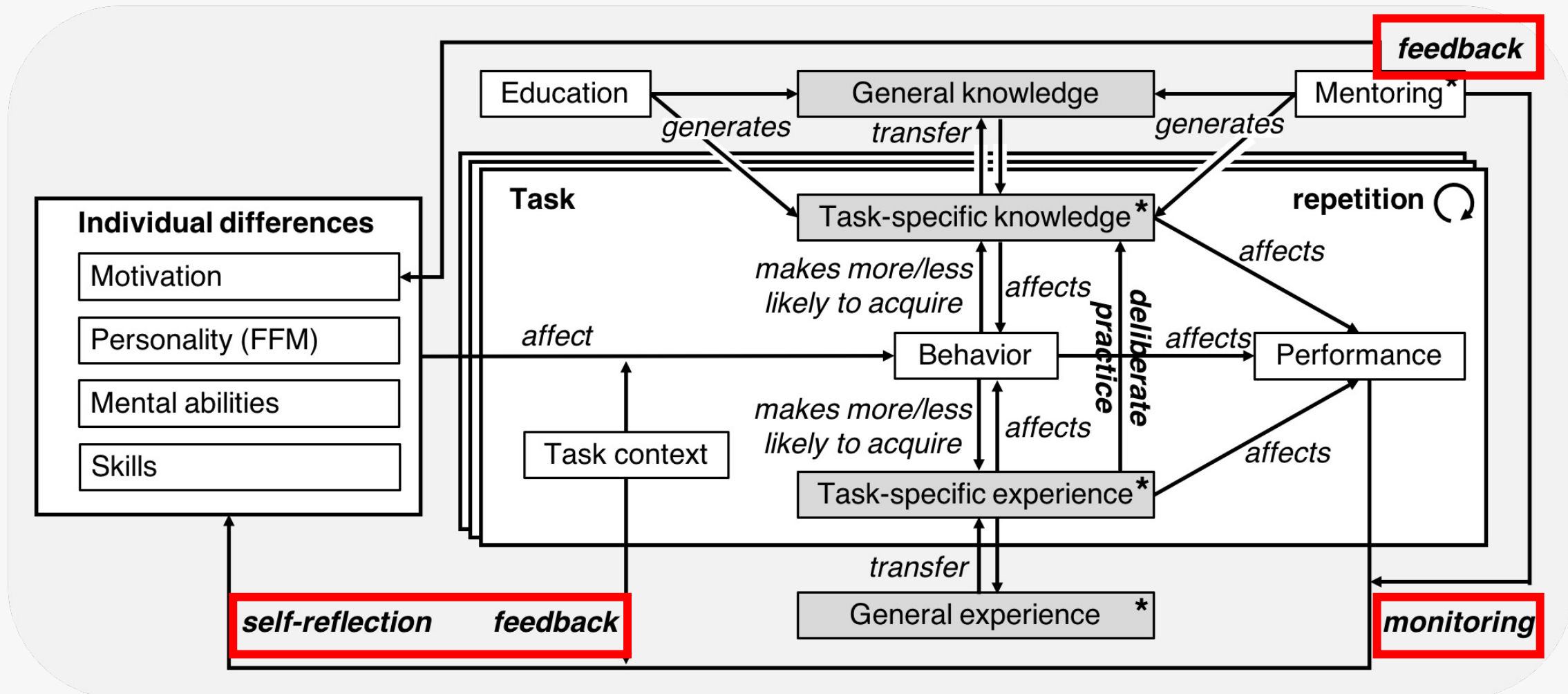
- Having **more experience** does not automatically lead to **better performance** (Ericsson et al., 1993)
- Performance may even **decrease** over time (Feltovich, 2006)
- Length of experience only weak correlate of job performance (Ericsson, 2006)
- Deliberate practice: „***Prolonged efforts to improve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

Deliberate Practice: Self-Reflection

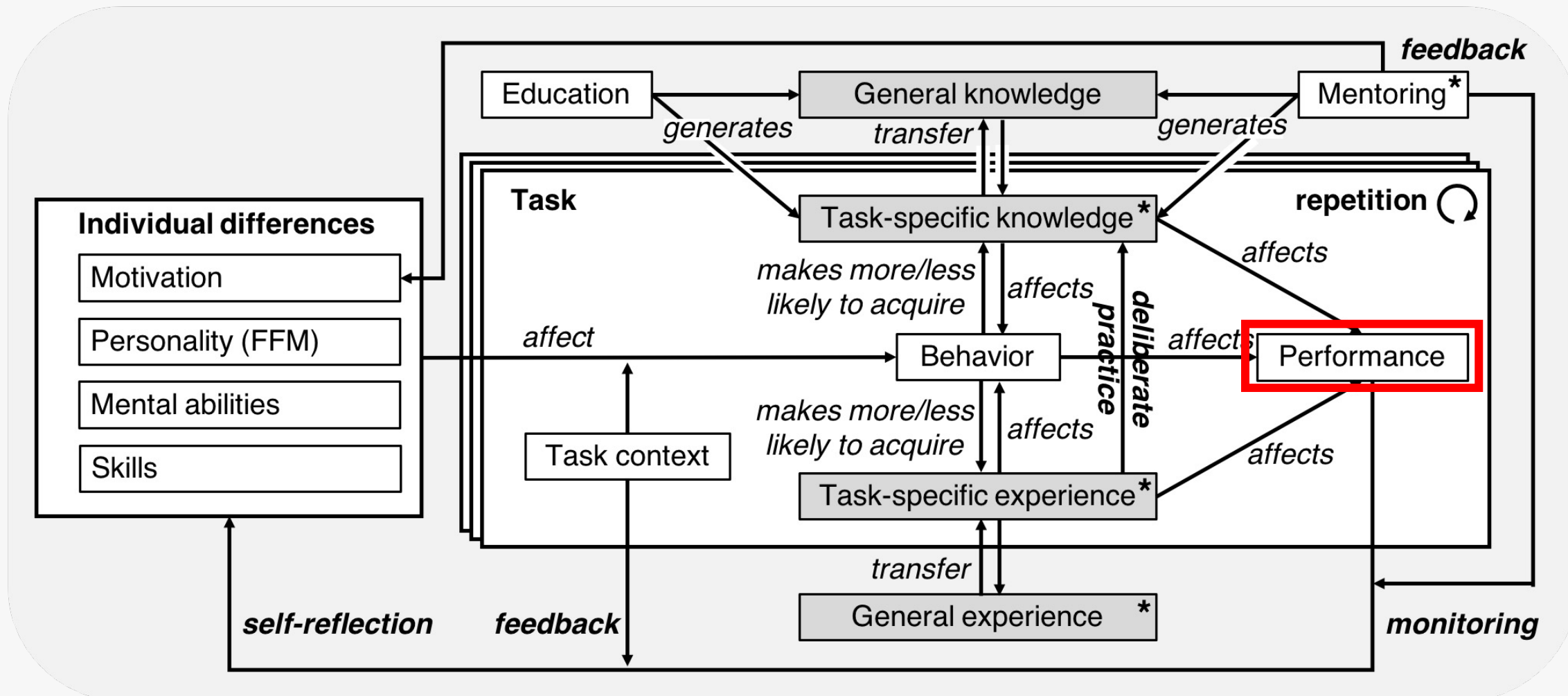


- **(Self-)reflection** and **feedback** important to **monitor** progress towards goal achievement (Locke and Latham, 1990)
- “[T]he more **channels of accurate and helpful feedback** we have access to, the better we are likely to perform.”
(Tourish and Hargie, 2003)
- **Mentors**, teachers, and peers are an important sources for feedback

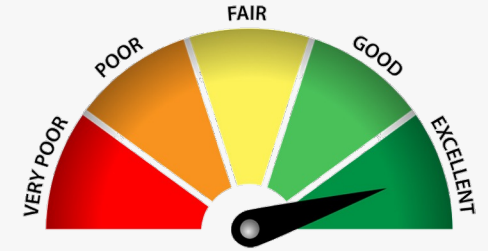
Final Conceptual Theory



Final Conceptual Theory



Performance



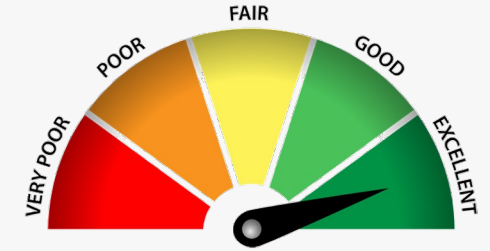
Scope of this work:

- We do **not** treat performance as a **dependent variable** that we try to explain 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.)

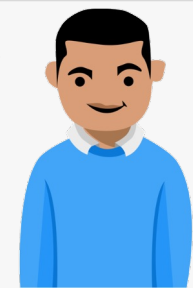
*„Everyone can write [...] code which a machine can read and process but the key lies in writing concise and understandable code which [...] **people who have never used that piece of code before [can read].**“*



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.**"*



How are experience and expertise related?



Experience vs. Expertise

- Self-assessment with **semantic differential** (novice to expert) and **Dreyfus expertise model**

Semantic Differential Scale

- Beginning of survey:

Please rate your Java programming expertise on the following scale:

1 (Novice)	2	3	4	5	6 (Expert)
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

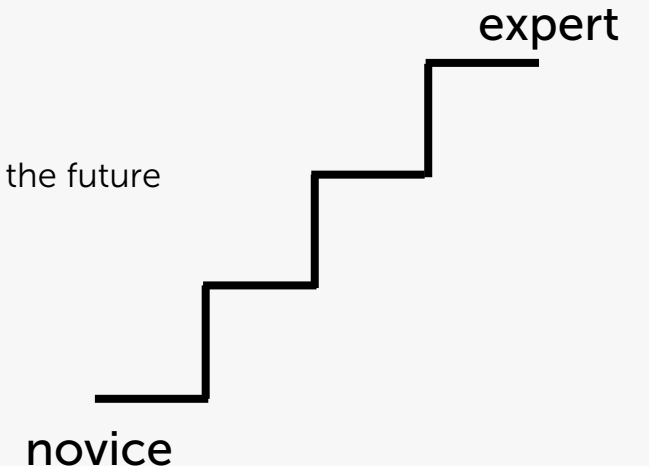
...

- End of survey:

Please rate your own Java programming expertise according to the five stages described below.

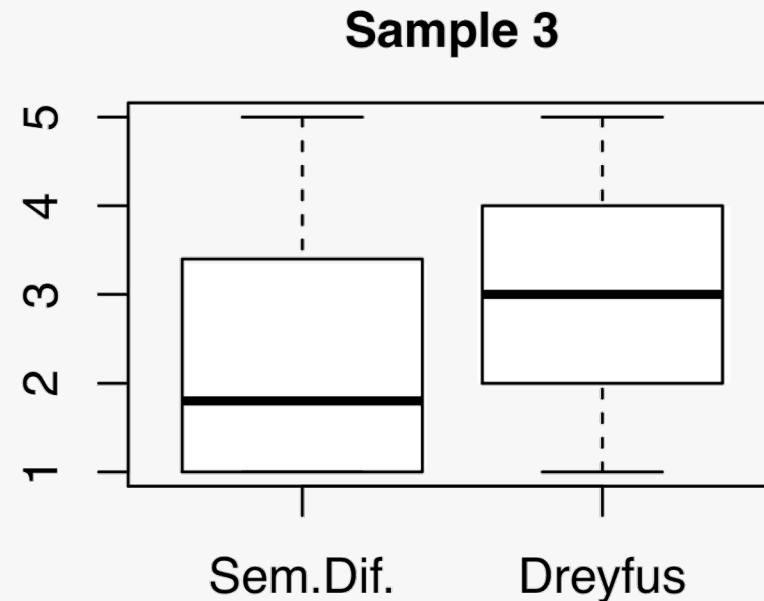
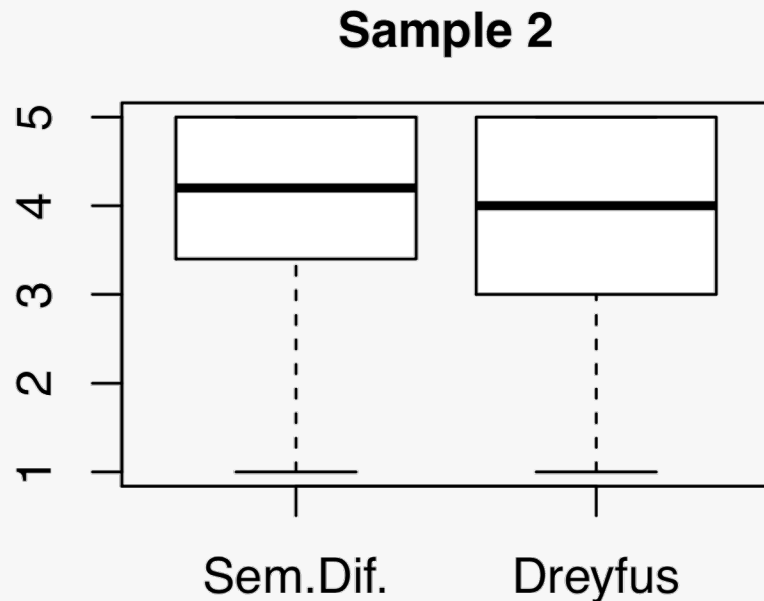
Discrete Expertise Model

- **Stage 1 (Novice):**
 - has little or no experience
 - wants unambiguous rules to accomplish his/her tasks
 - is able to handle small, isolated tasks
- **Stage 2 (Advanced Beginner):**
 - has gained some experience
 - can work more independently than a novice
 - knows general principles in a limited context, but does not have a holistic understanding ("big picture")
- **Stage 3 (Competent):**
 - has a holistic understanding of the problem domain
 - bases his/her work on deliberate planning and extensive past experience
 - can apply general maxims (e.g. design patterns) easily to specific contexts
- **Stage 4 (Proficient):**
 - has a vast amount of experience that he/she can intuitively apply to new contexts
 - can easily differentiate between irrelevant and important details
 - constantly reflects on what he/she has done and revises own approach to perform better in the future
- **Stage 5 (Expert):**
 - he/she is a major source of knowledge and information for others
 - primarily works from his/her intuition



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

- 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

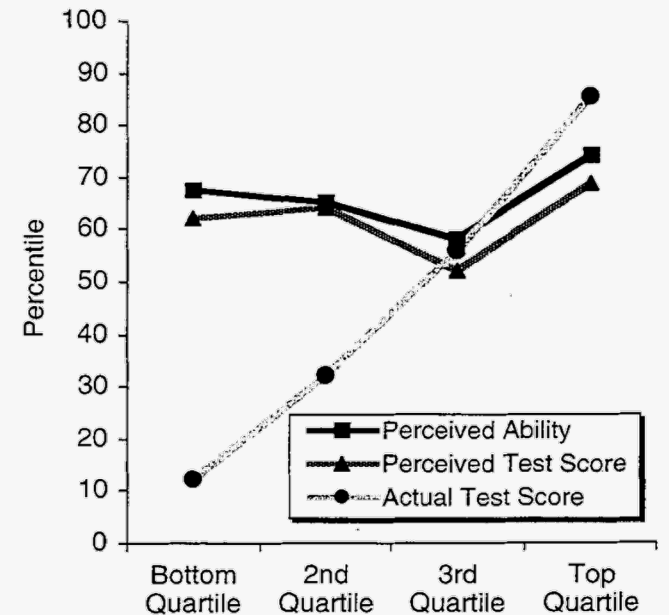
Journal of Personality and Social Psychology
1999, Vol. 77, No. 6, 1121–1134

Copyright 1999 by the American Psychological Association, Inc.
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.





Takeaways

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

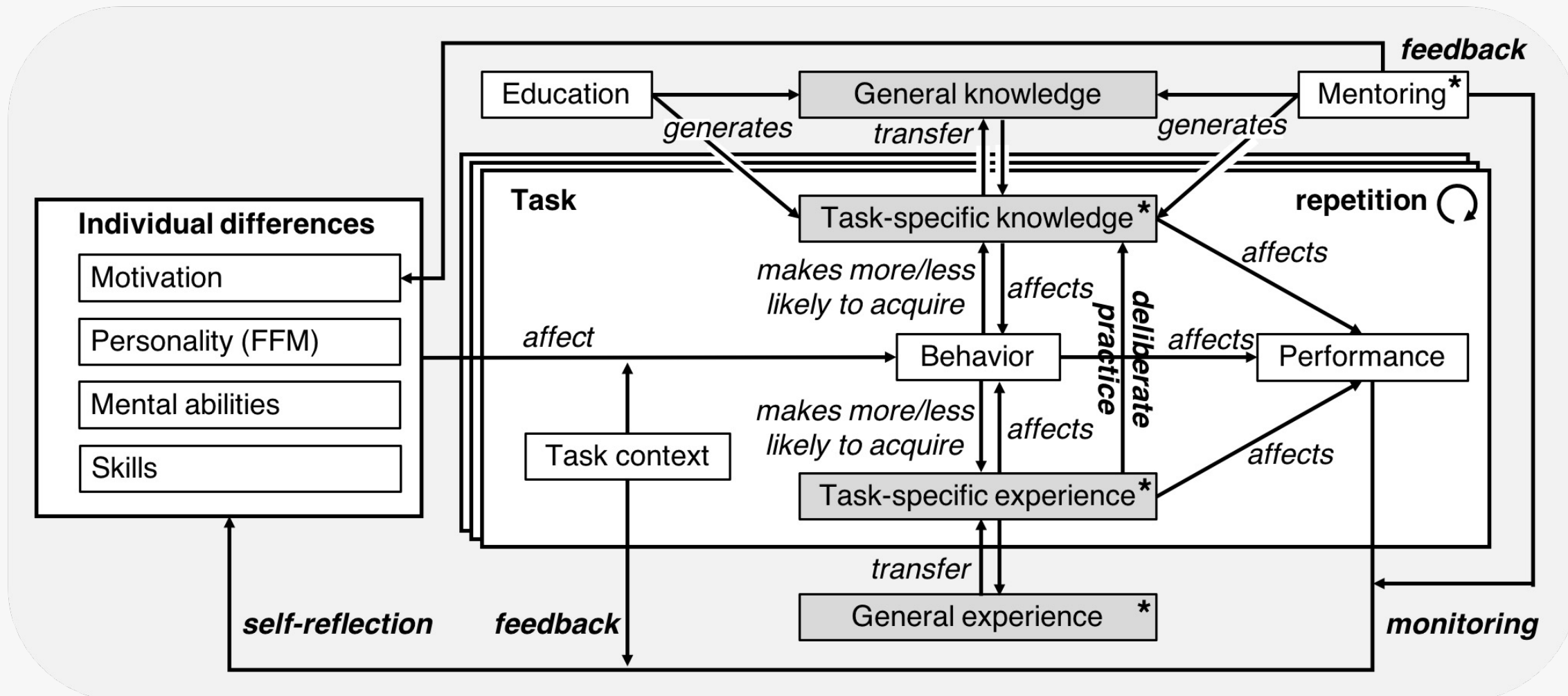


Summary for Employers

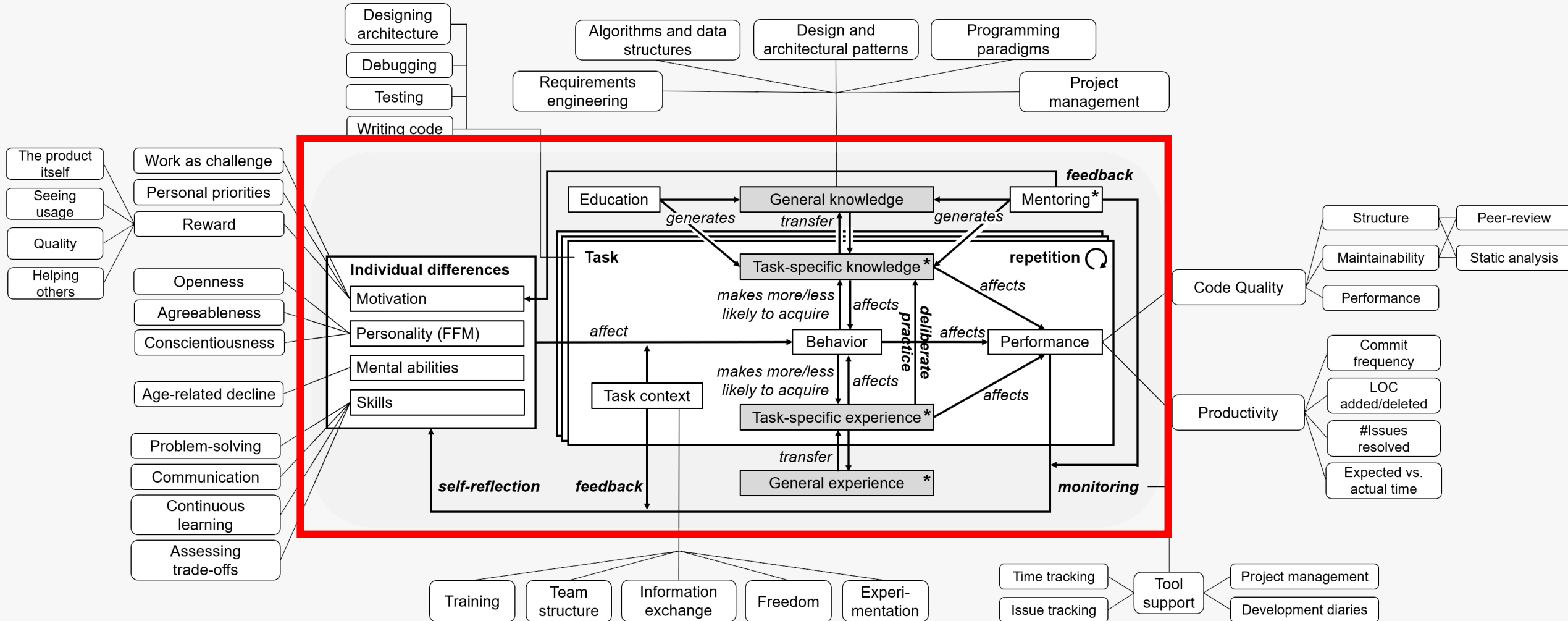
- Learn what **(de)motivates** their employees:
 - Main motivation: problem solving
 - Main demotivation: non-challenging work
- Ideas on how to build supportive work environment **supporting self-improvement** of staff:
 - Good mix of continuity and change in software development process
 - Communicate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

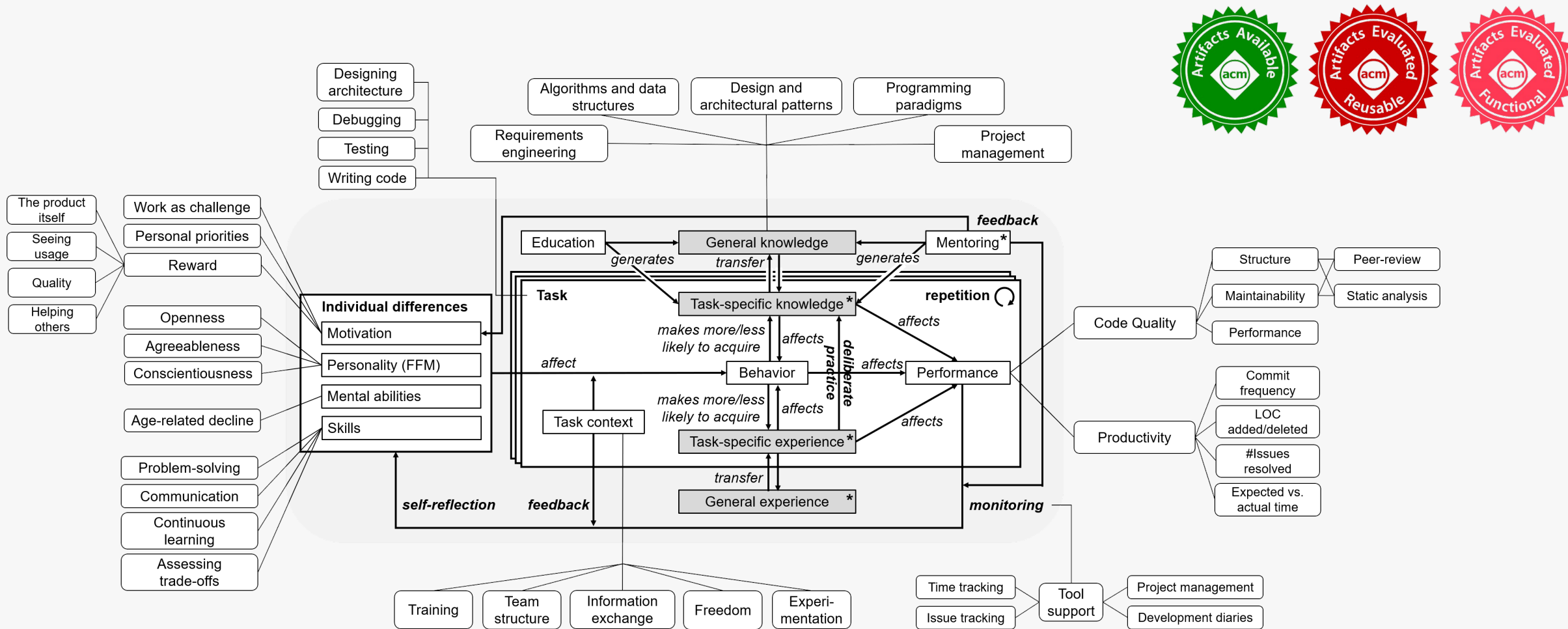


Core of Conceptual Theory



Complete Conceptual Theory





Sebastian Baltes
 @s_baltes

expertise.sbaltes.com
Data and scripts available on Zenodo