# Expertise in Software Development

Towards an Interdisciplinary Theory

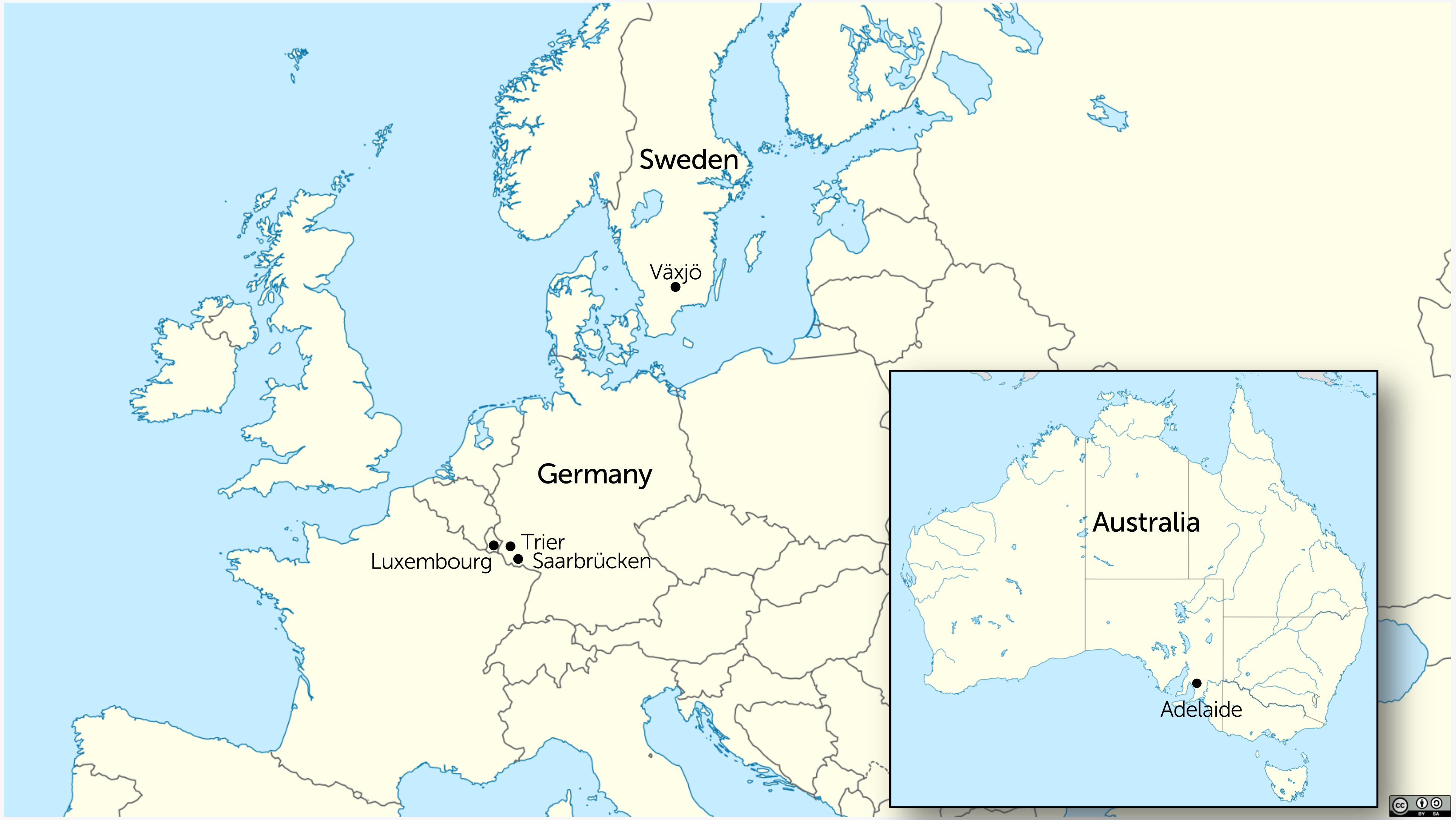
**Sebastian Baltes**

@s_baltes

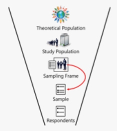
empirical-software.engineering

THE UNIVERSITY of ADELAIDE

## My Background



# Evidence-based Practice through Practice-based Evidence



Issues in Sampling Software Developers

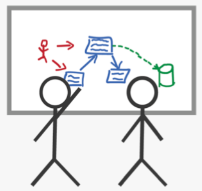
Methodology

airbnb

Constructing Urban Tourism Space Digitally

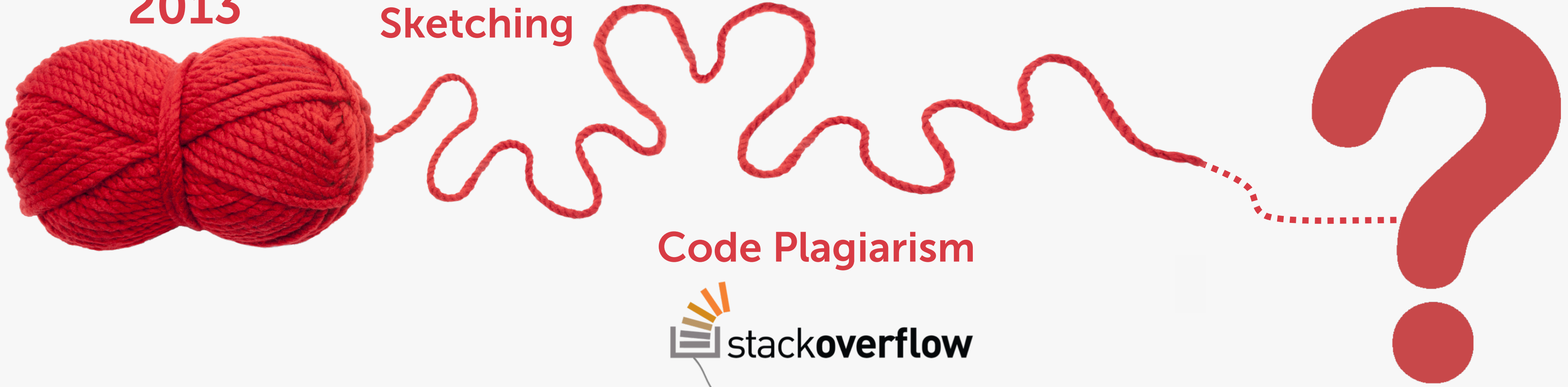
Interdisciplinary Research

Expertise Development



Sketching

2013



Code Plagiarism

stackoverflow

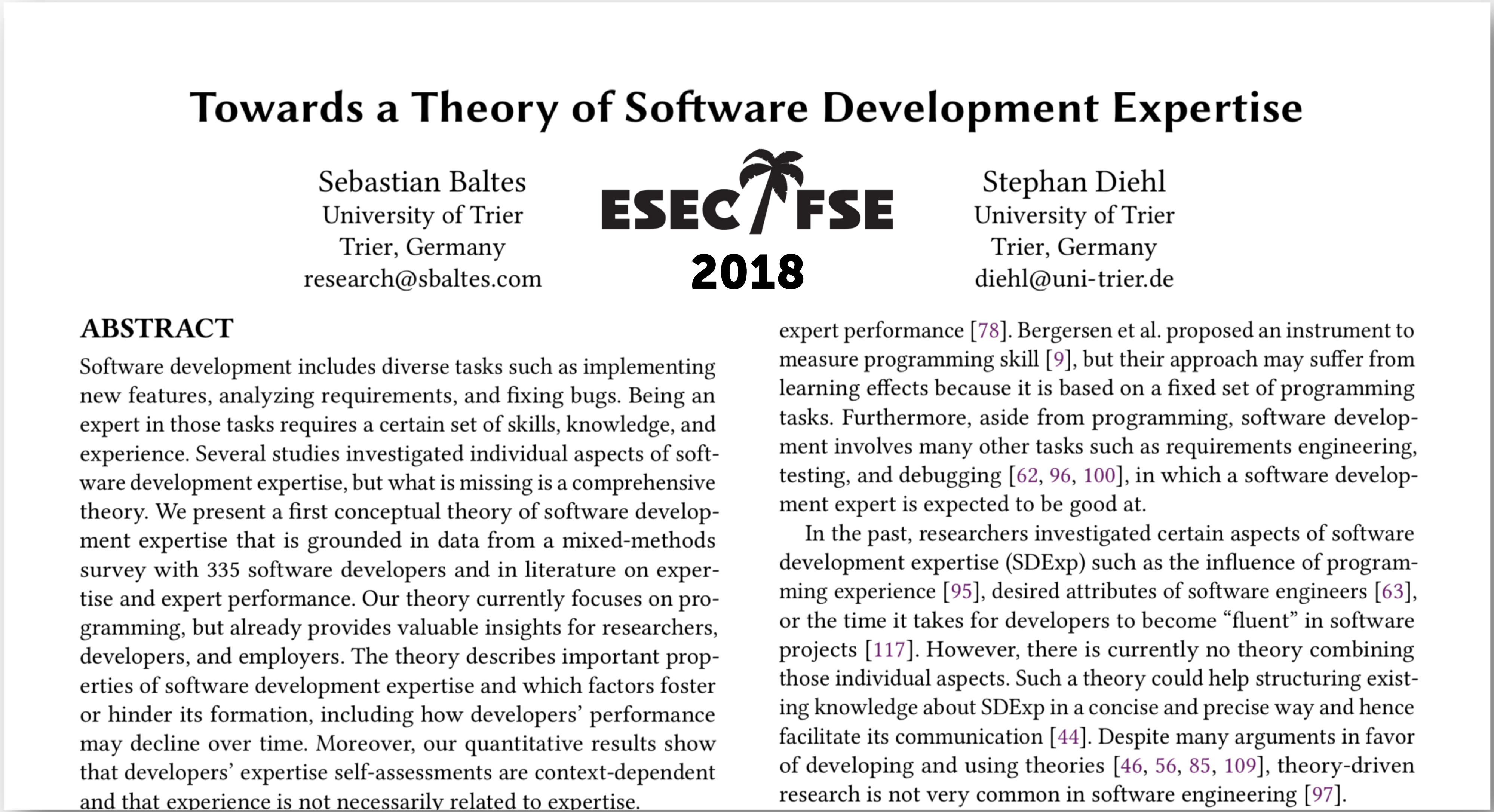
Regular Expressions

RegViz

Continuous Integration

GitHub ⟷ Travis CI

## 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

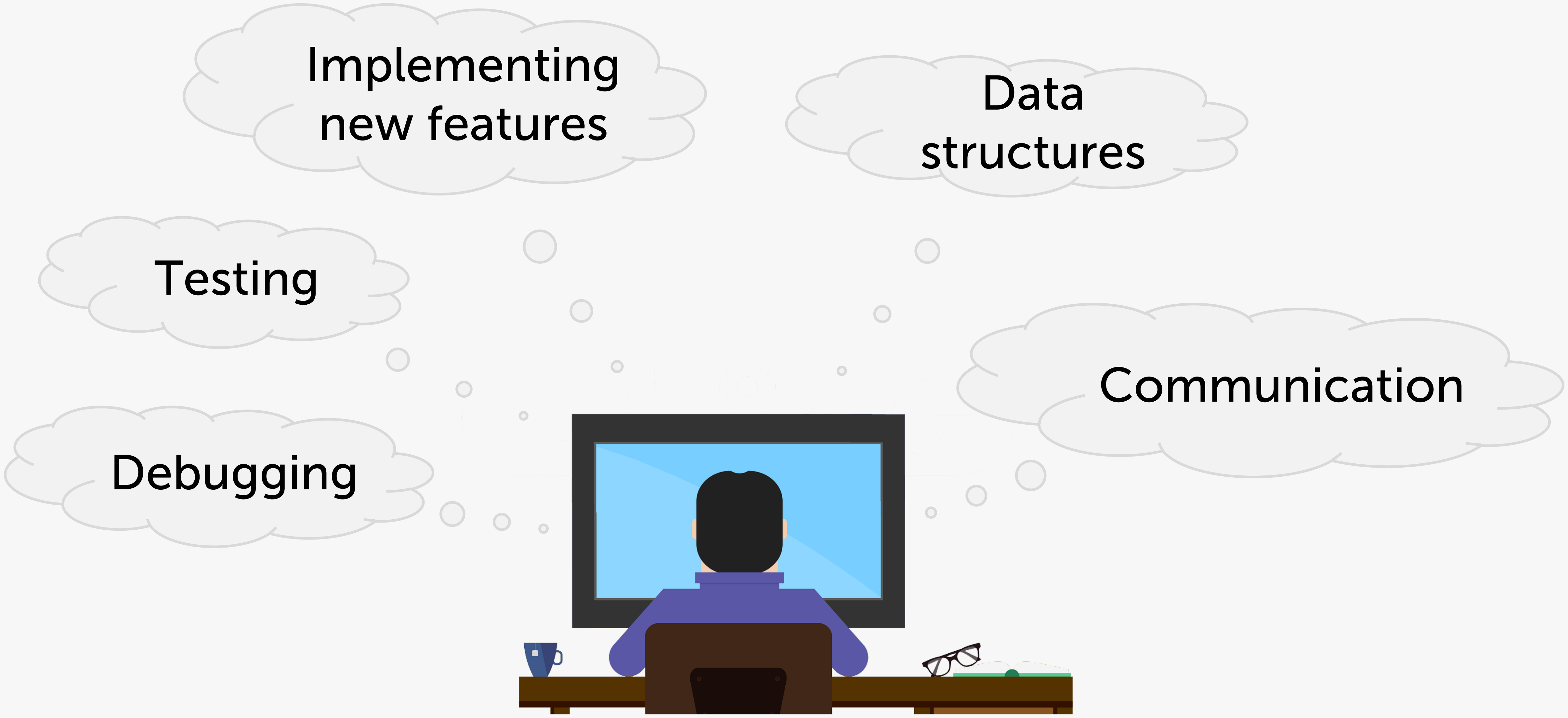
Software development includes diverse tasks such as implementing new features, analyzing requirements, and fixing bugs. Being an expert in those tasks requires a certain set of skills, knowledge, and experience. Several studies investigated individual aspects of software development expertise, but what 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.

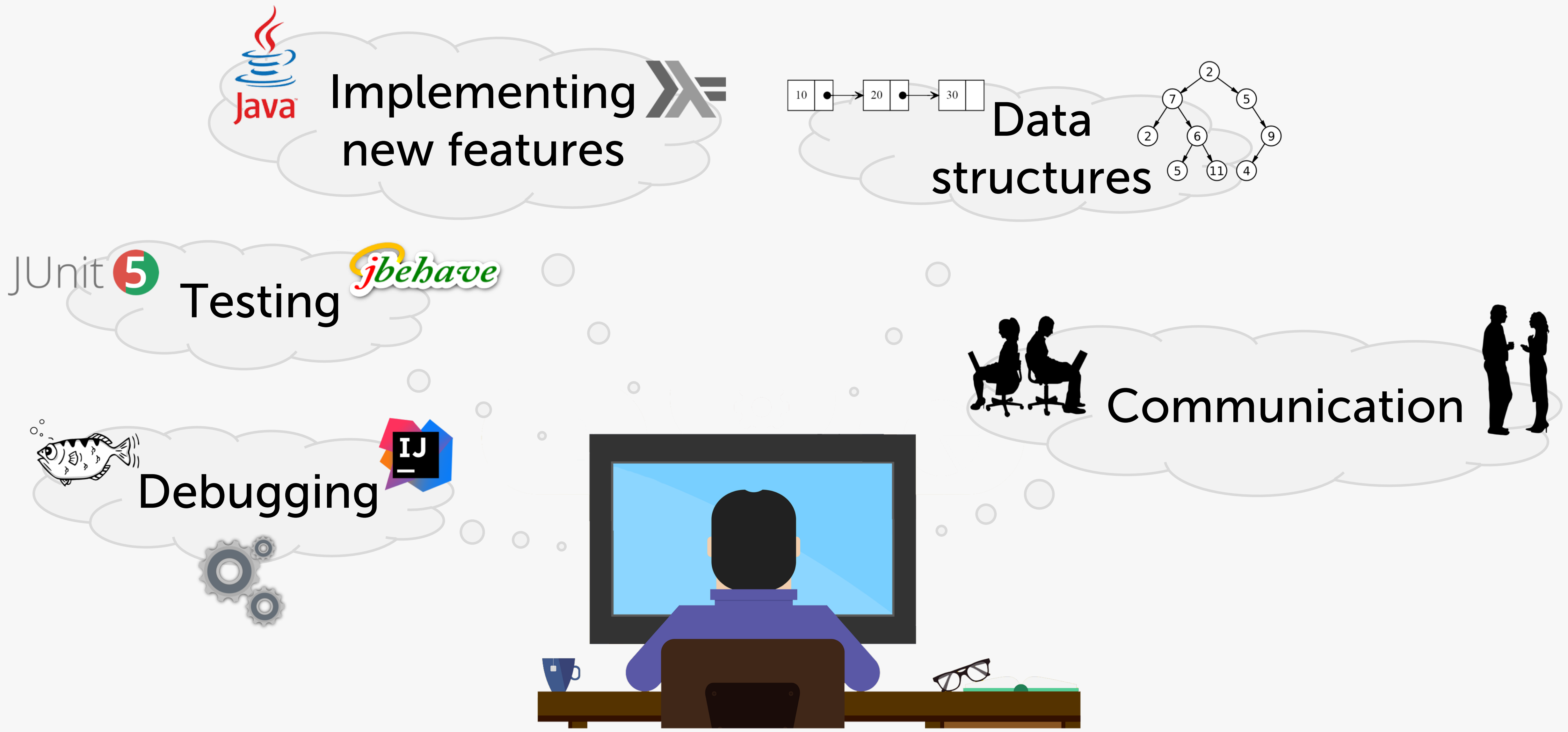
In the past, researchers investigated certain aspects of software development expertise (SDExp) such as the influence of programming experience [95], desired attributes of software engineers [63], or the time it takes for developers to become "fluent" in software projects [117]. However, there is currently 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?



# Software Development Expertise?



## How to structure all those expertise-related aspects?

## Which factors influence expertise development over time?

## How are experience and expertise related?

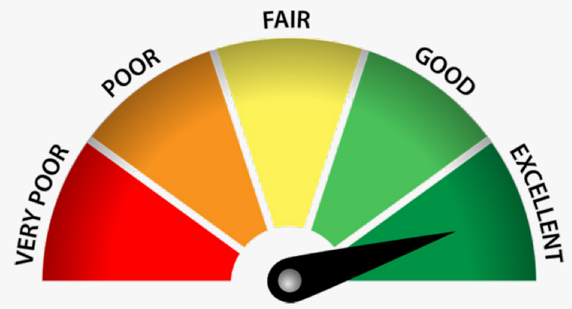
## Definitions

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

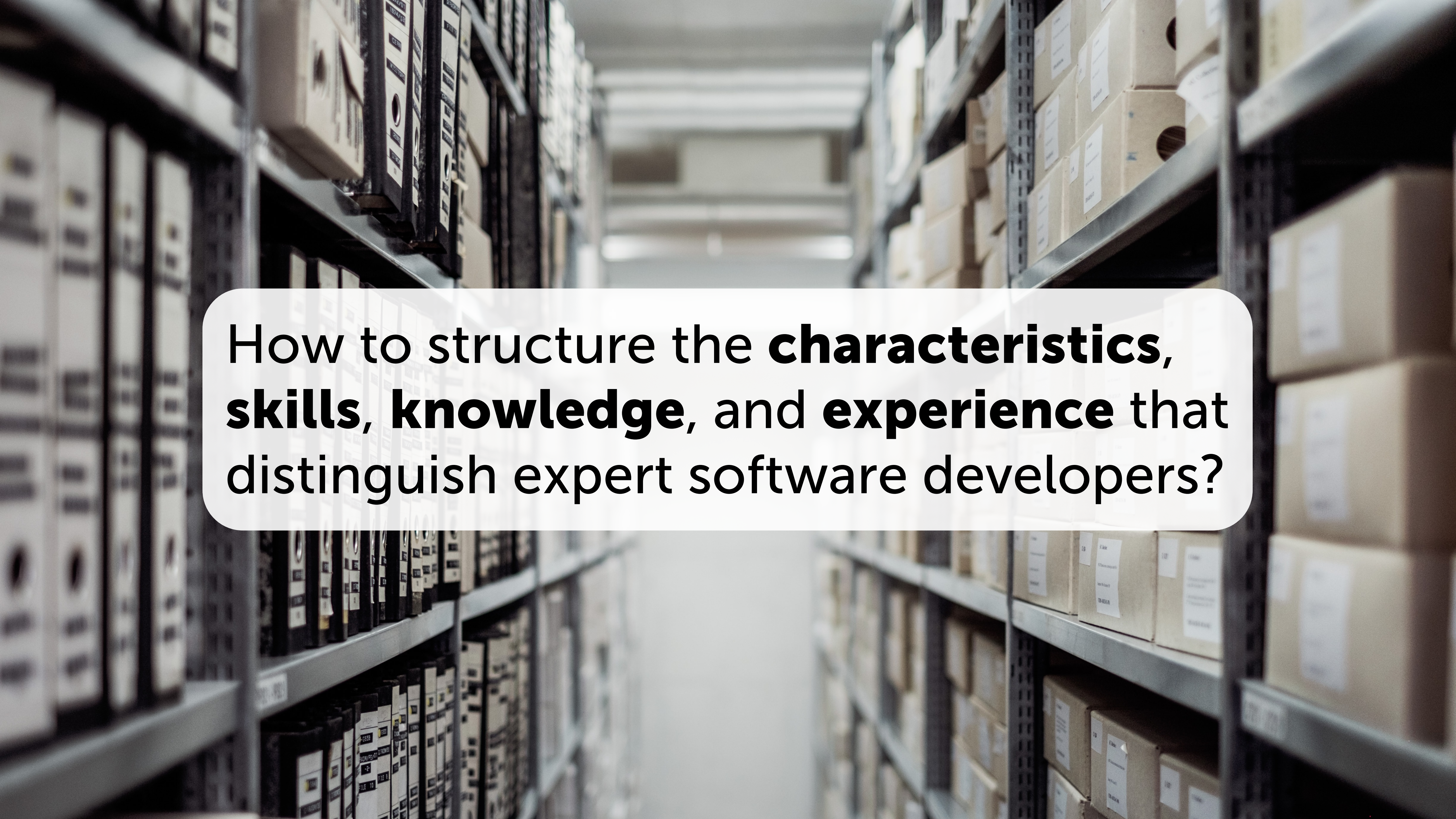
# Expert Performance



- In some areas (e.g., chess), there exist **representative tasks** and **objective criteria** for identifying experts

- Software development includes **many different tasks**

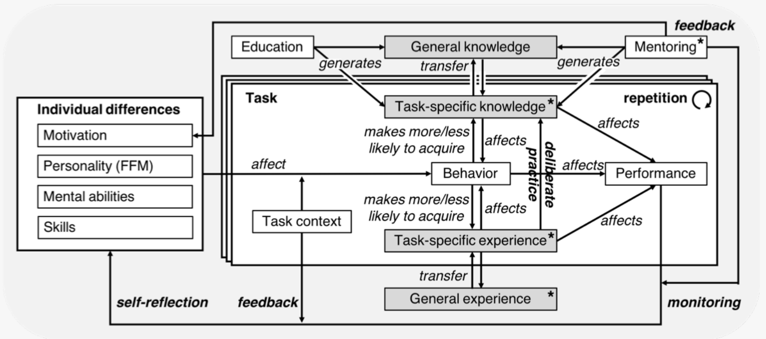
- Much more **difficult** to find objective measures for quantifying software development expert performance

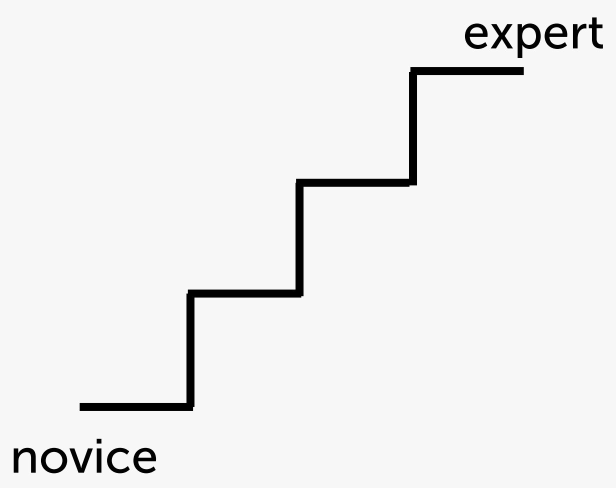


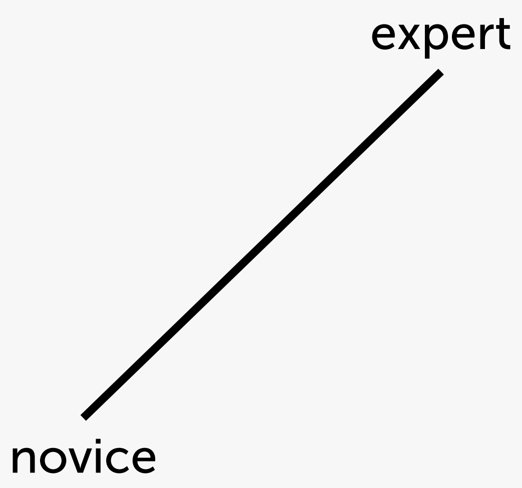
How to structure the **characteristics**, **skills**, **knowledge**, and **experience** that distinguish expert software developers?

# 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







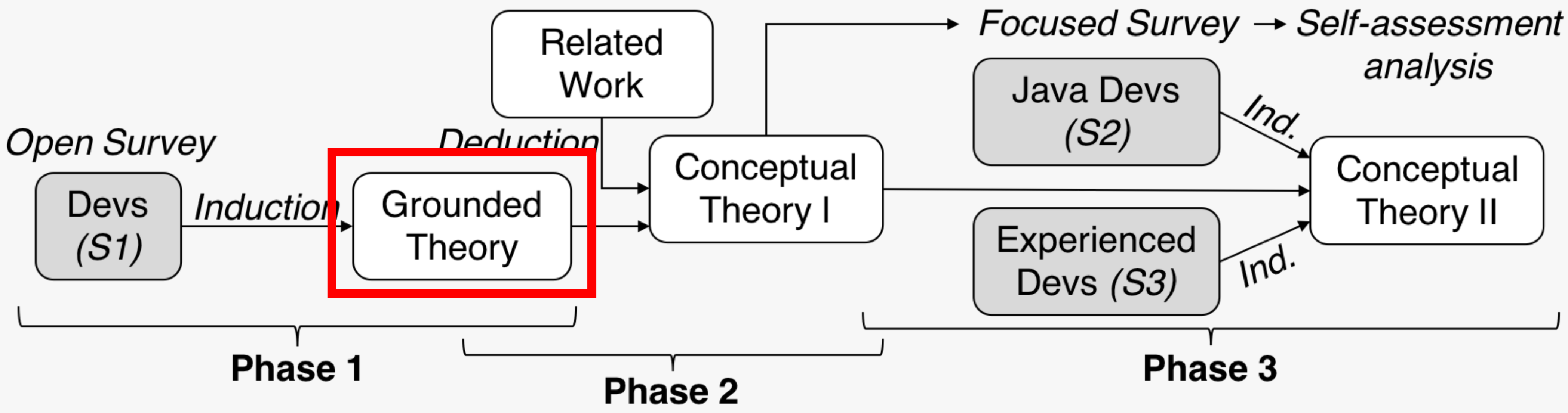
## Theory Classification

- A **process theory** intends to explain and understand *"how an entity changes and develops"* over time (Ralph, 2018)

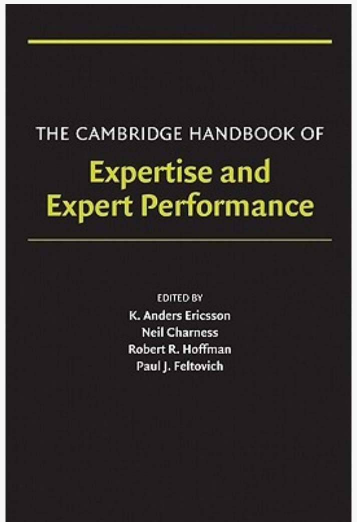
- In a **teleological process theory**, an entity *"constructs an envisioned end state, takes action to reach it, and monitors the progress"* (van de Ven and Poole, 1995)

- **Our theory:**
  - *Entity:*
    Individual software developer working on different software development tasks
  - *Envisioned end state:*
    Being an expert in (some of) those tasks

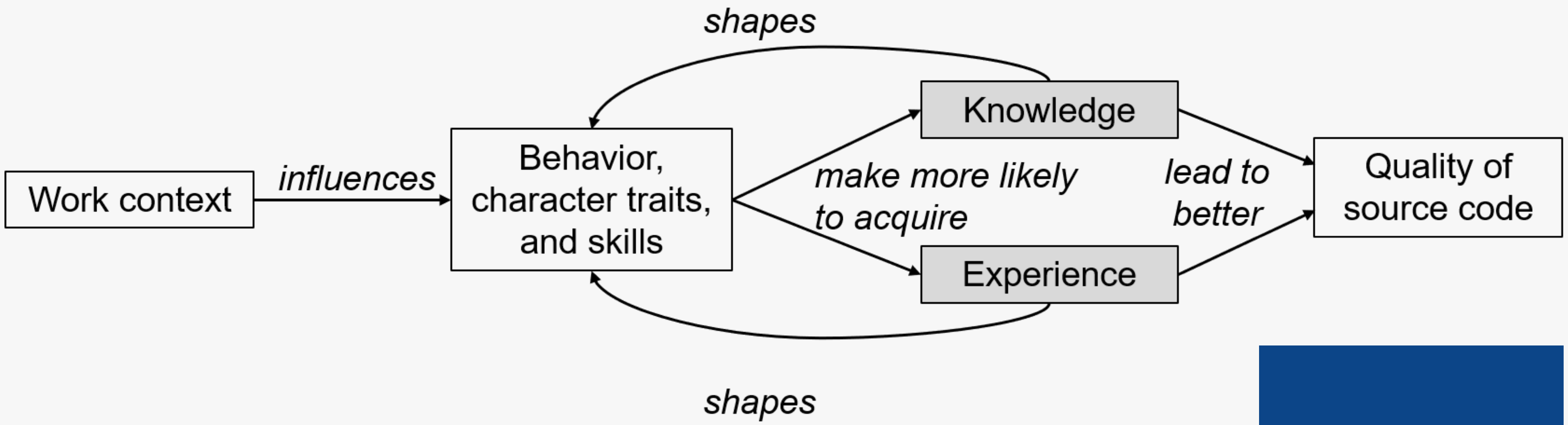
# Research Design

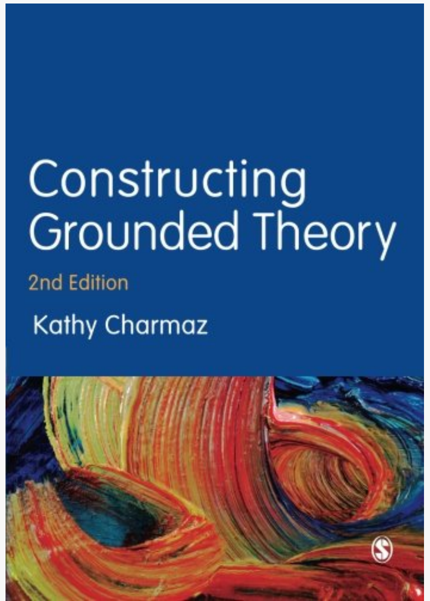


- **Induction:** 335 online survey participants in total
- **Deduction:** Main source *"Cambridge Handbook of Expertise and Expert Performance"*

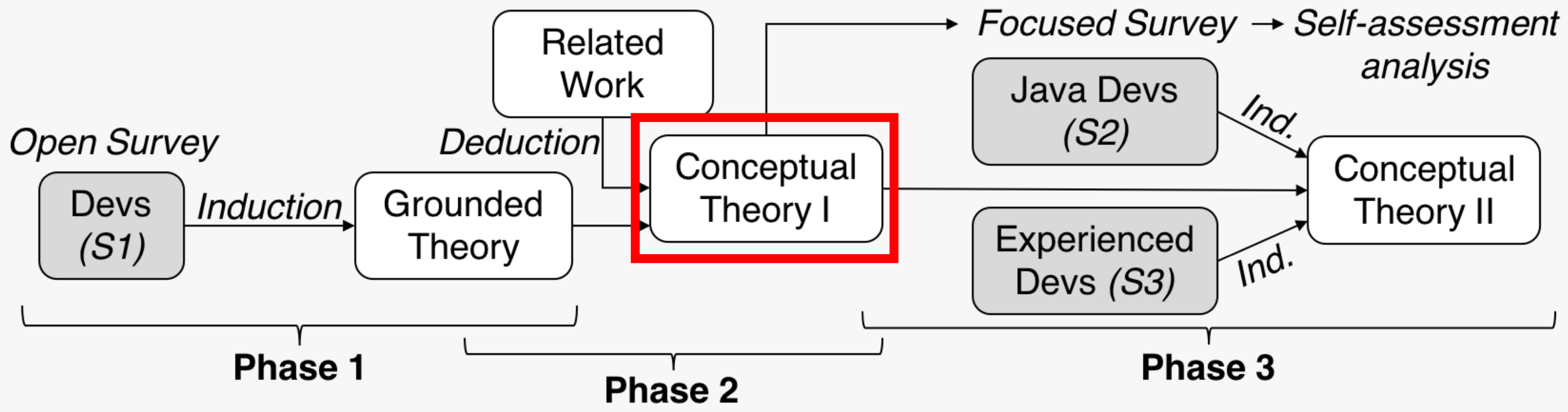


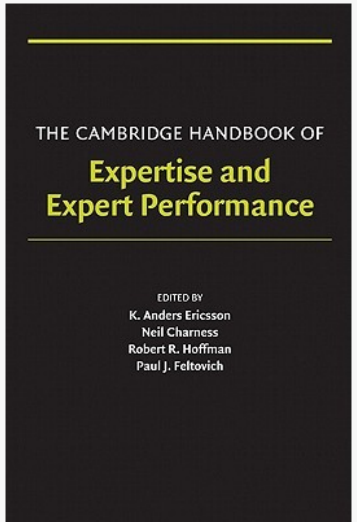
# Grounded Theory





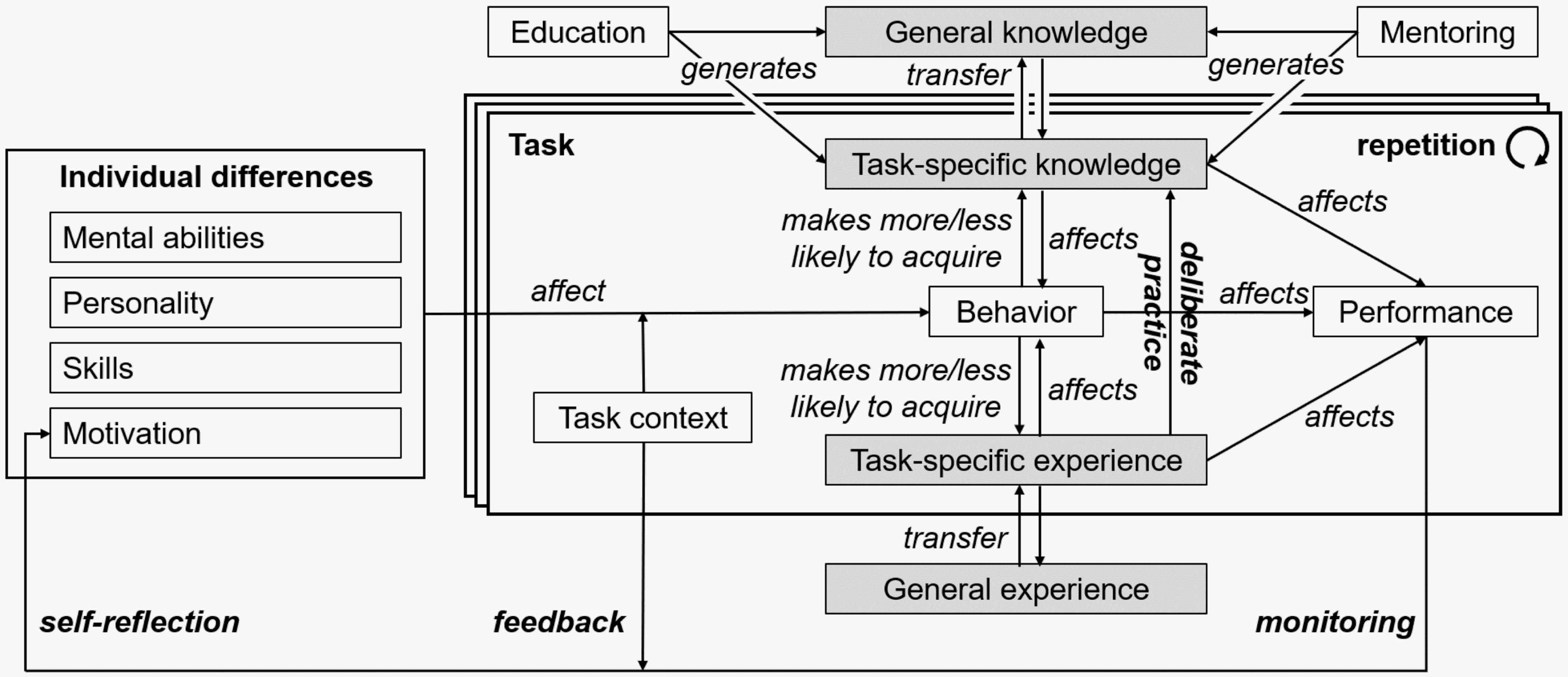
# Research Design



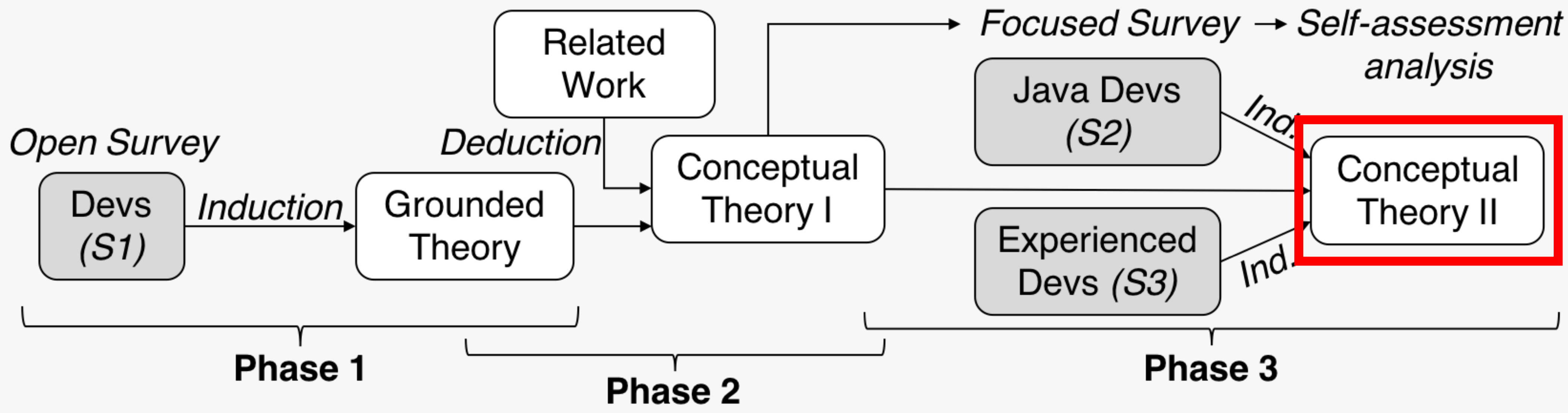


- **Induction:** 335 online survey participants in total
- **Deduction:** Main source *"Cambridge Handbook of Expertise and Expert Performance"*

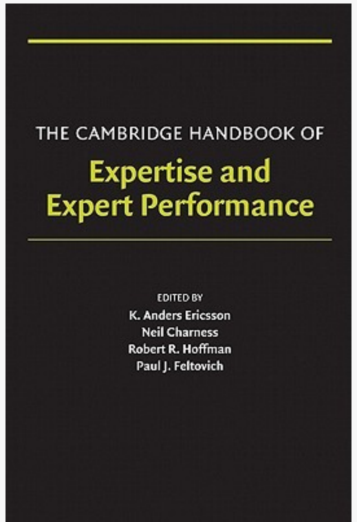
# Preliminary Conceptual Theory



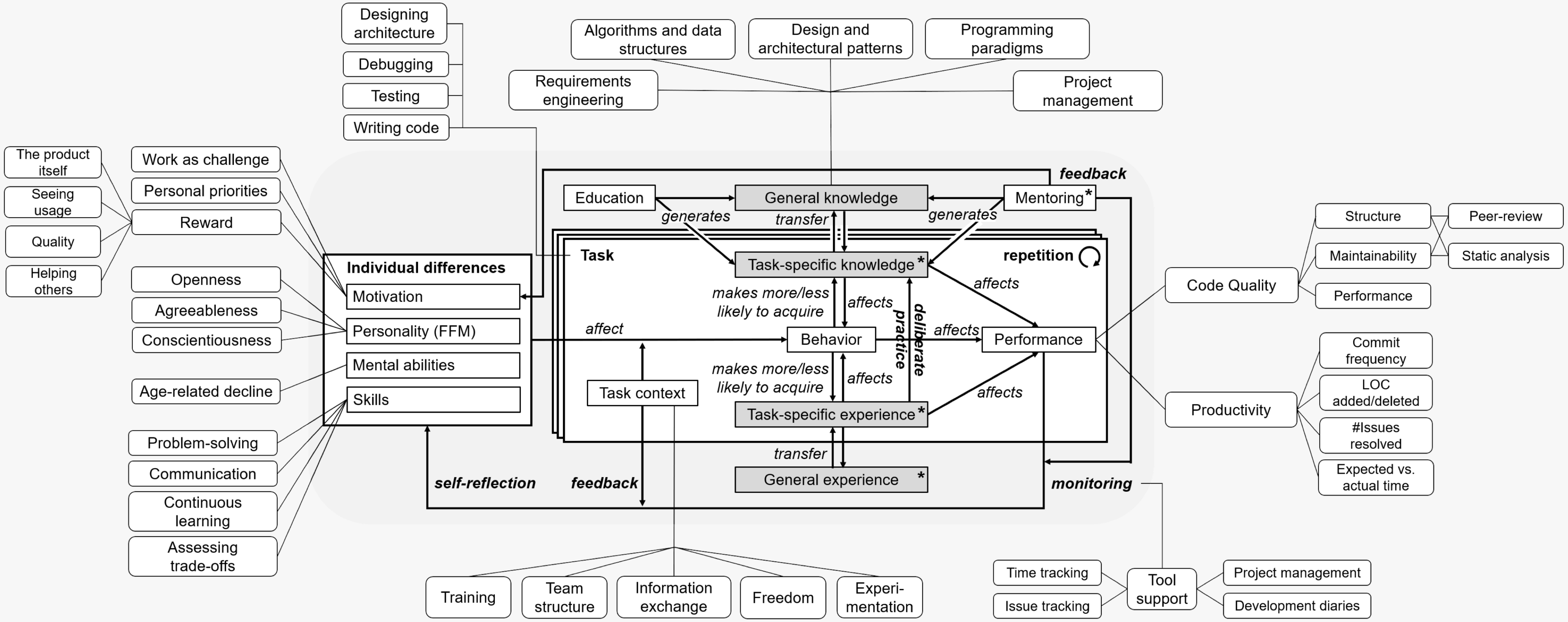
# Research Design



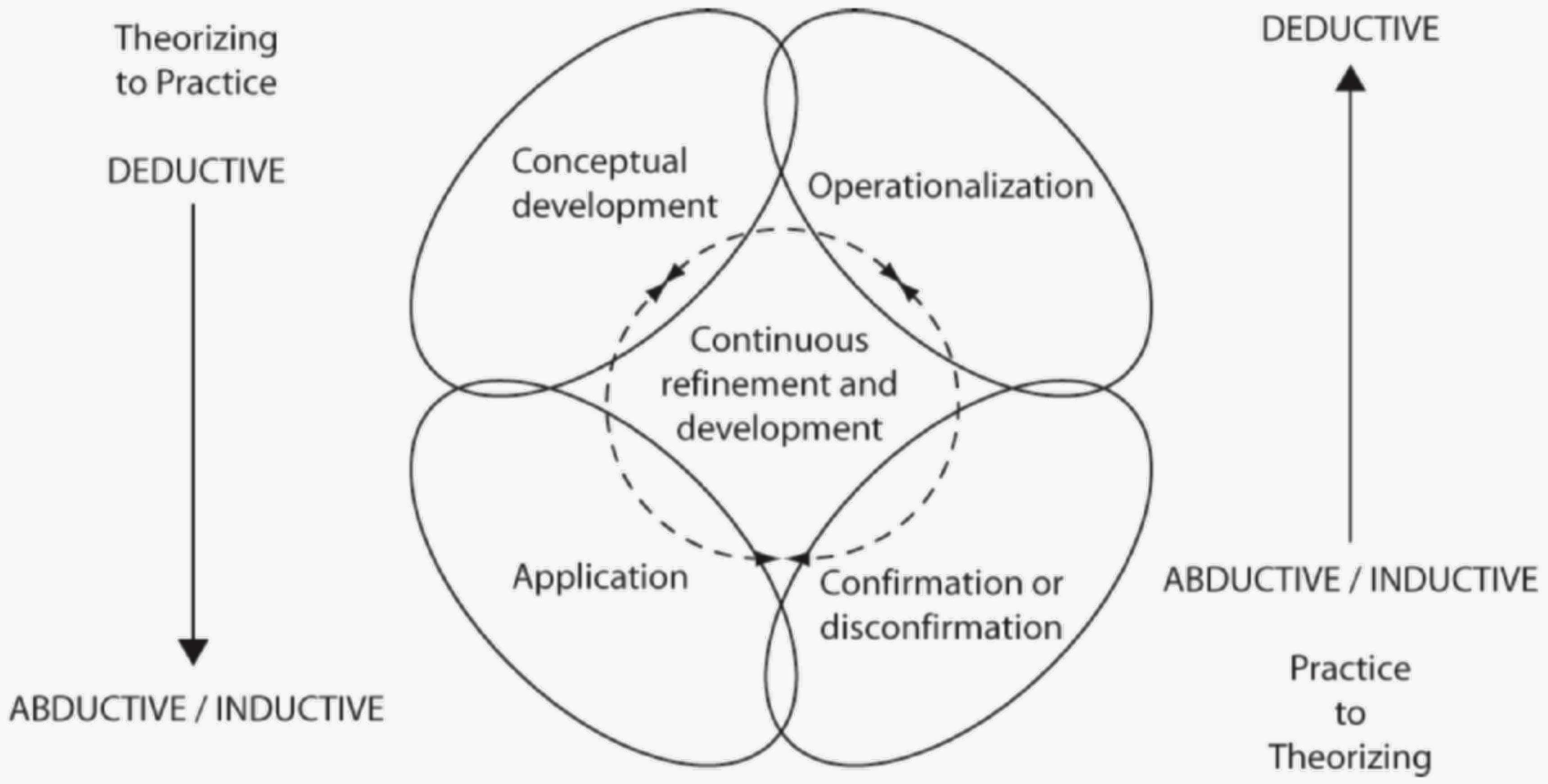
- **Induction:** 335 online survey participants in total
- **Deduction:** Main source *"Cambridge Handbook of Expertise and Expert Performance"*



# Final Conceptual Theory

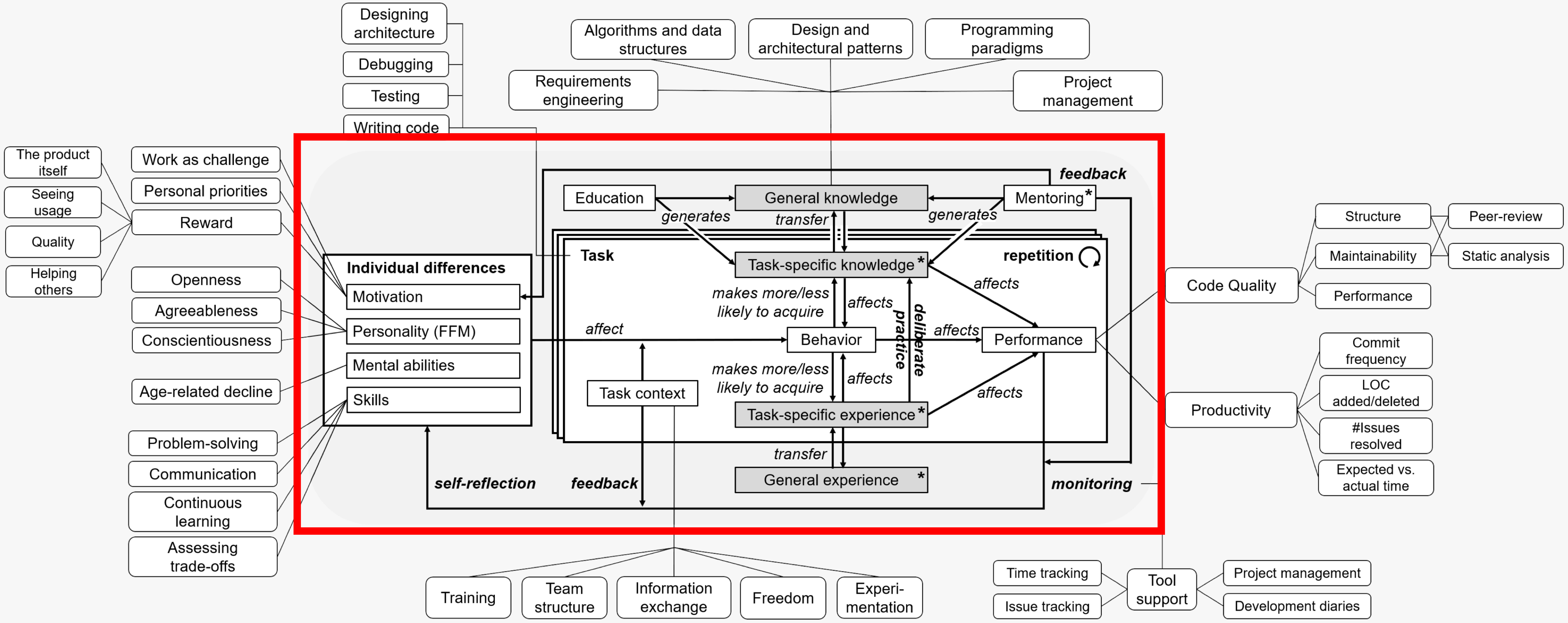


## Conceptual Theory?

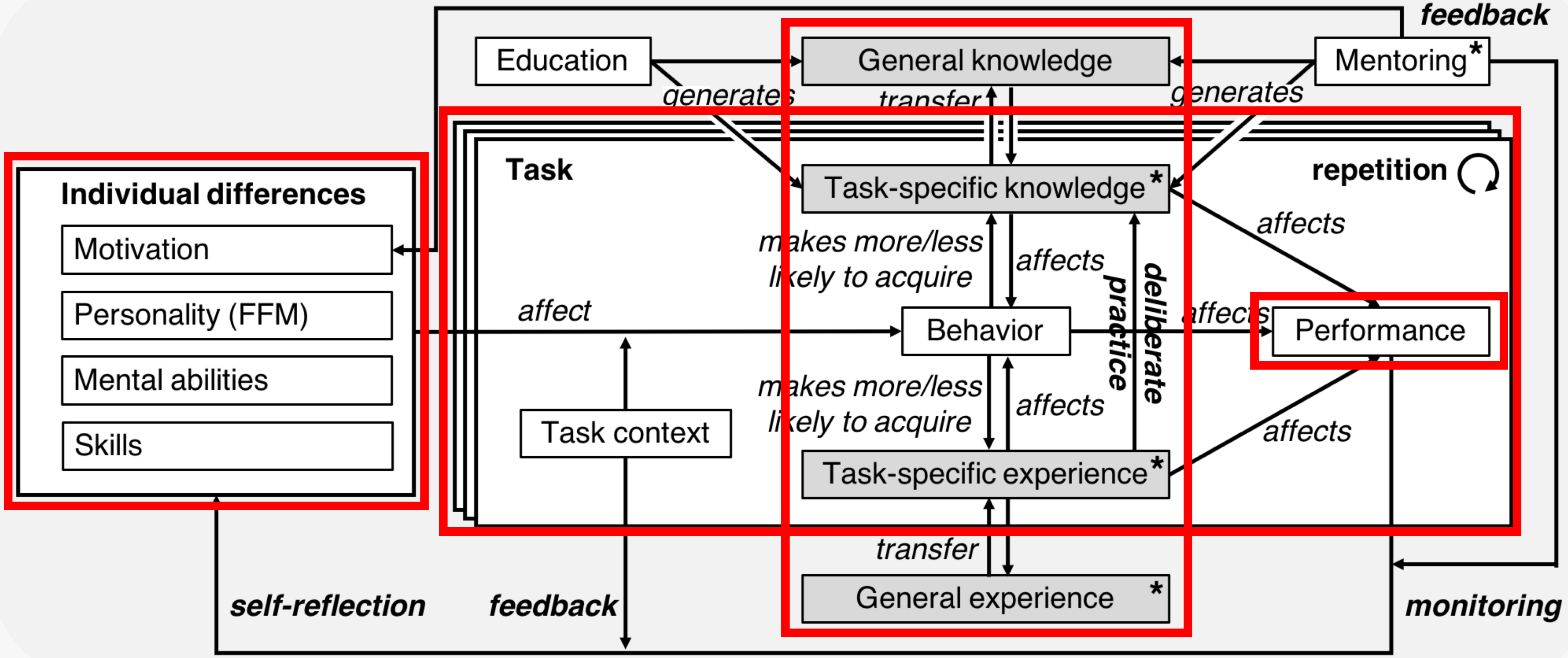


Building Theories in Software Engineering
Dag I.K. Sjøberg, Tore Dybå, Bente C.D. Anda, and Jo E. Hannay
in *Guide to Advanced Empirical Software Engineering* (2008).

# Final Conceptual Theory



# Final Conceptual Theory

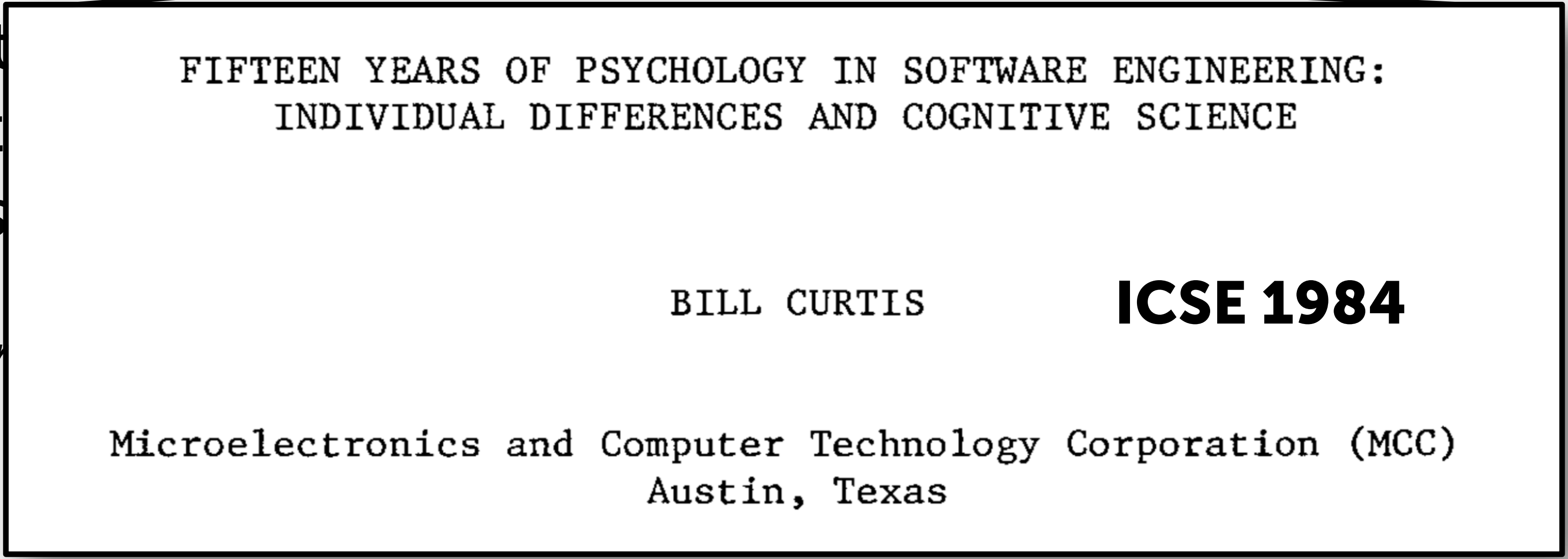


## Knowledge

- **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. *"syr*


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

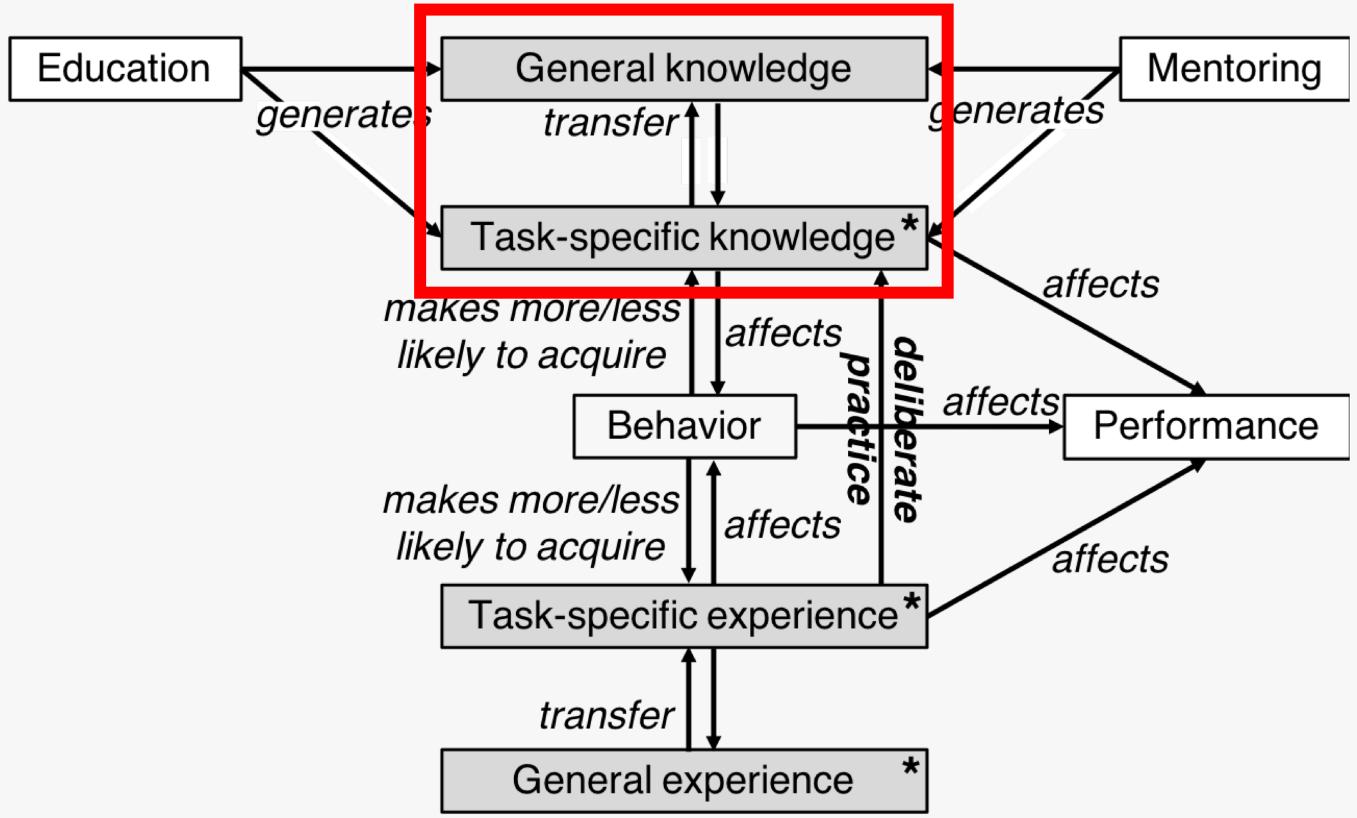
BILL CURTIS

**ICSE 1984**

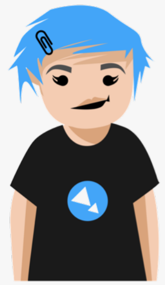
Microelectronics and Computer Technology Corporation (MCC)
Austin, Texas

# Knowledge

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



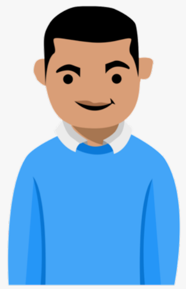
An *"intimate knowledge of the design and philosophy of the language"*



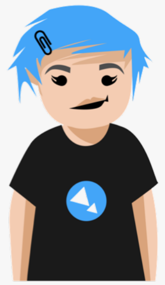
# Experience

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

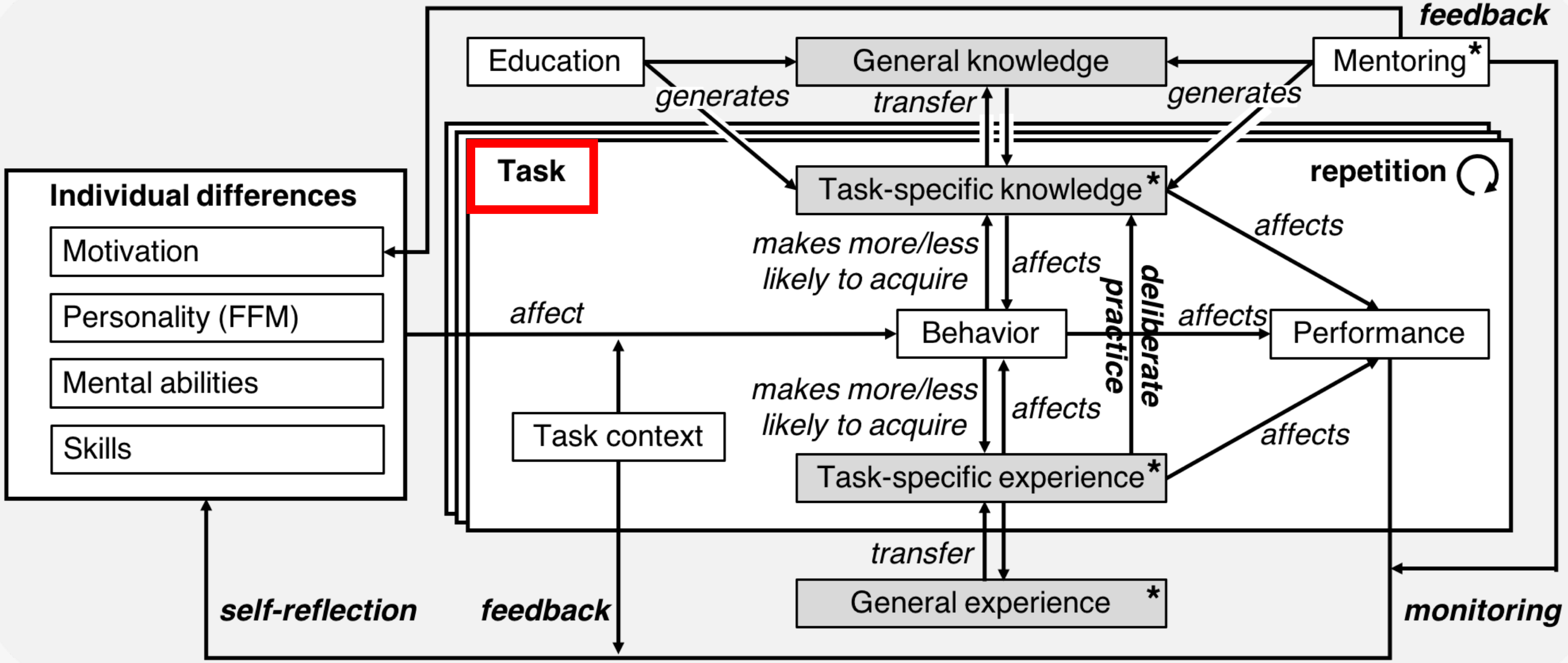
Having built „*everything from small projects to enterprise projects*“



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



# Final Conceptual Theory



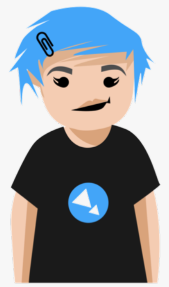
# Tasks

- Asked participants to name the **three most important tasks** that a software development expert should be good at

- Most frequently mentioned:
  1. Designing a software architecture
  2. Writing source code
  3. Analyzing and understanding requirements

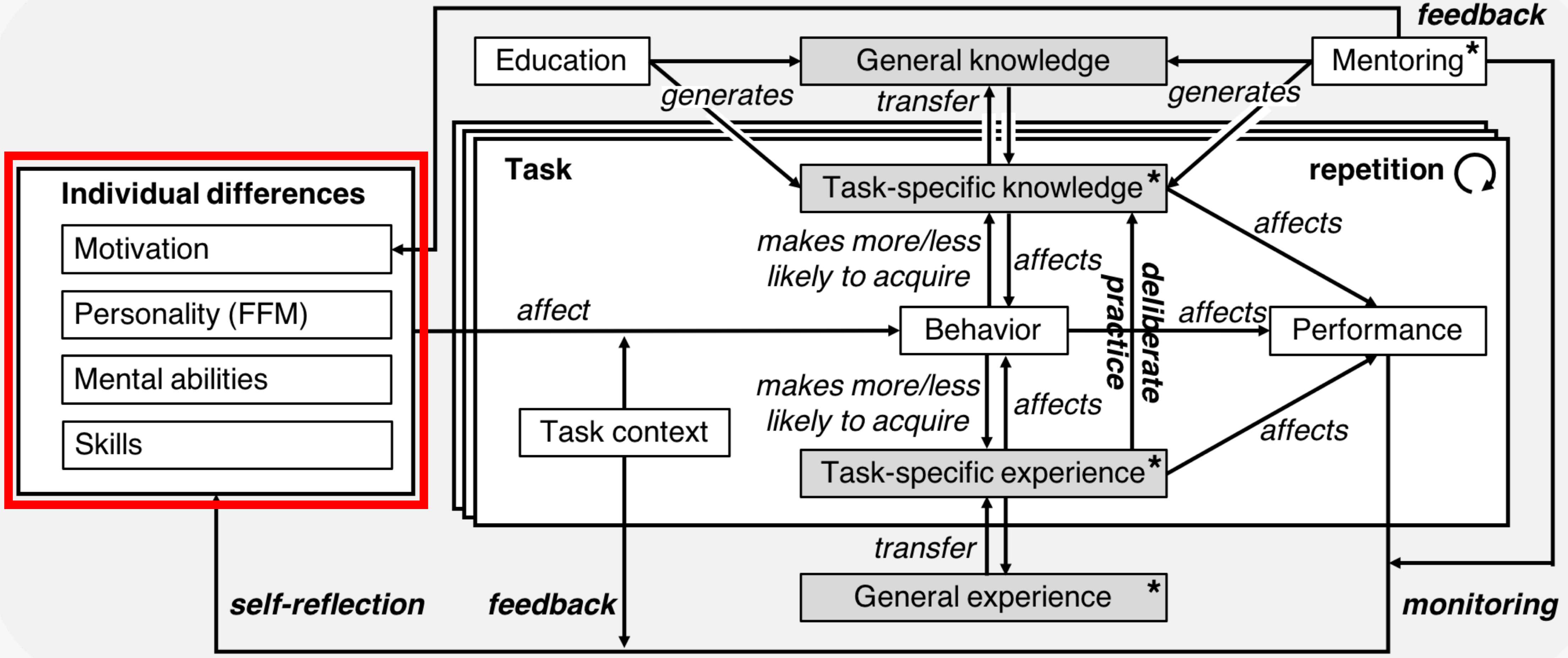
- Other mentioned tasks: testing, communicating, debugging

*"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

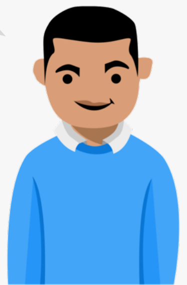


# Individual Differences: Motivation

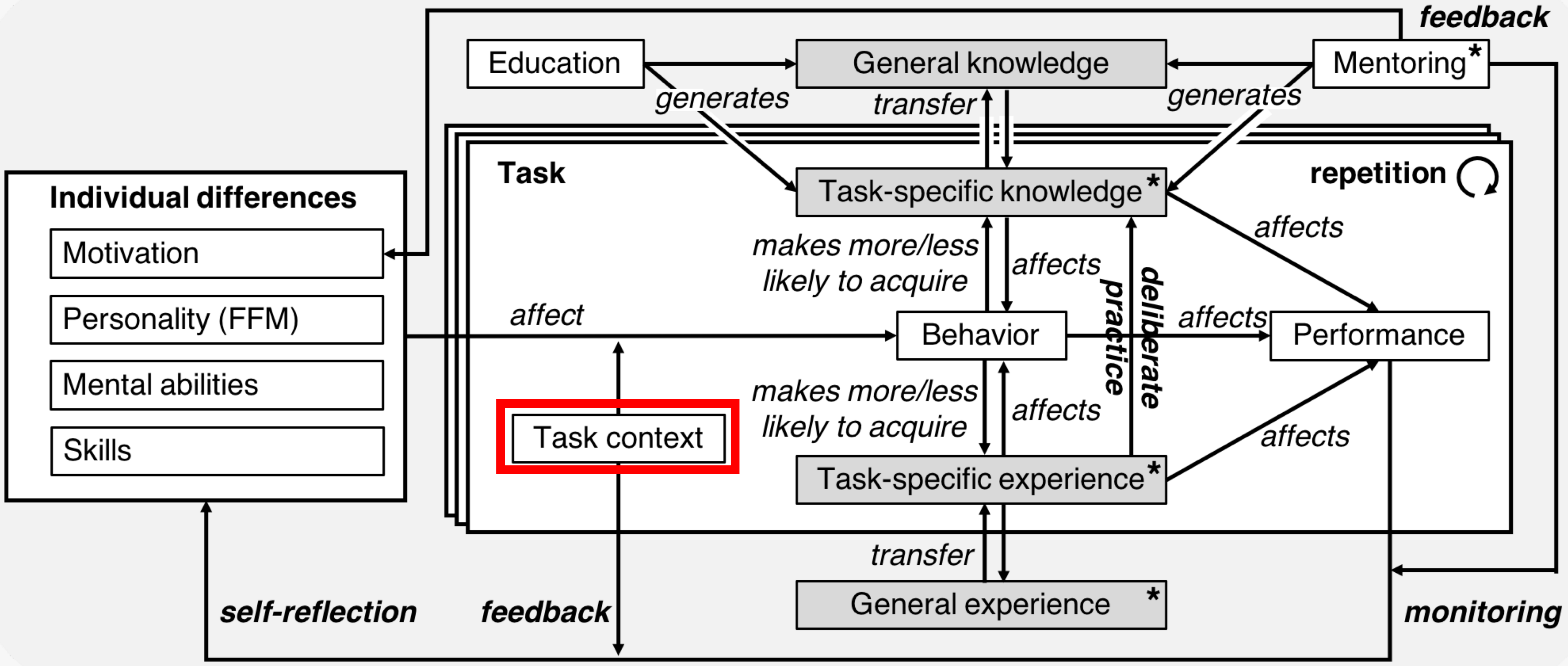
- Related work describes how **individual differences** affect expertise development
- Mental abilities and personality are relatively stable
- **Motivation can change** over time

- 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****."*



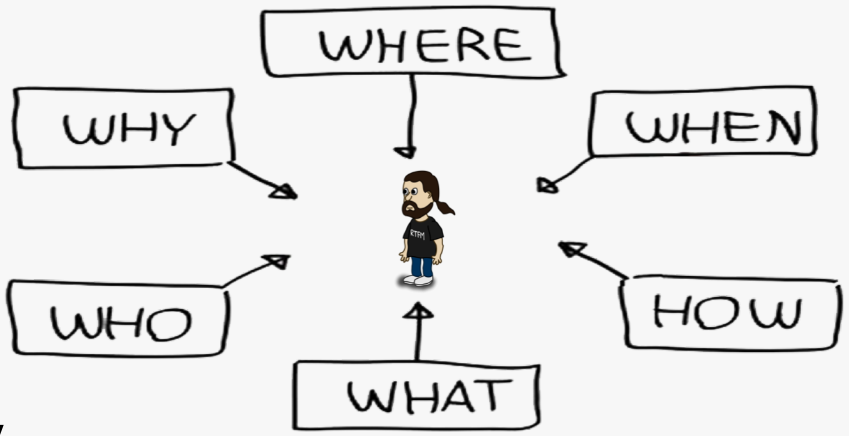
# Final Conceptual Theory



# Task Context

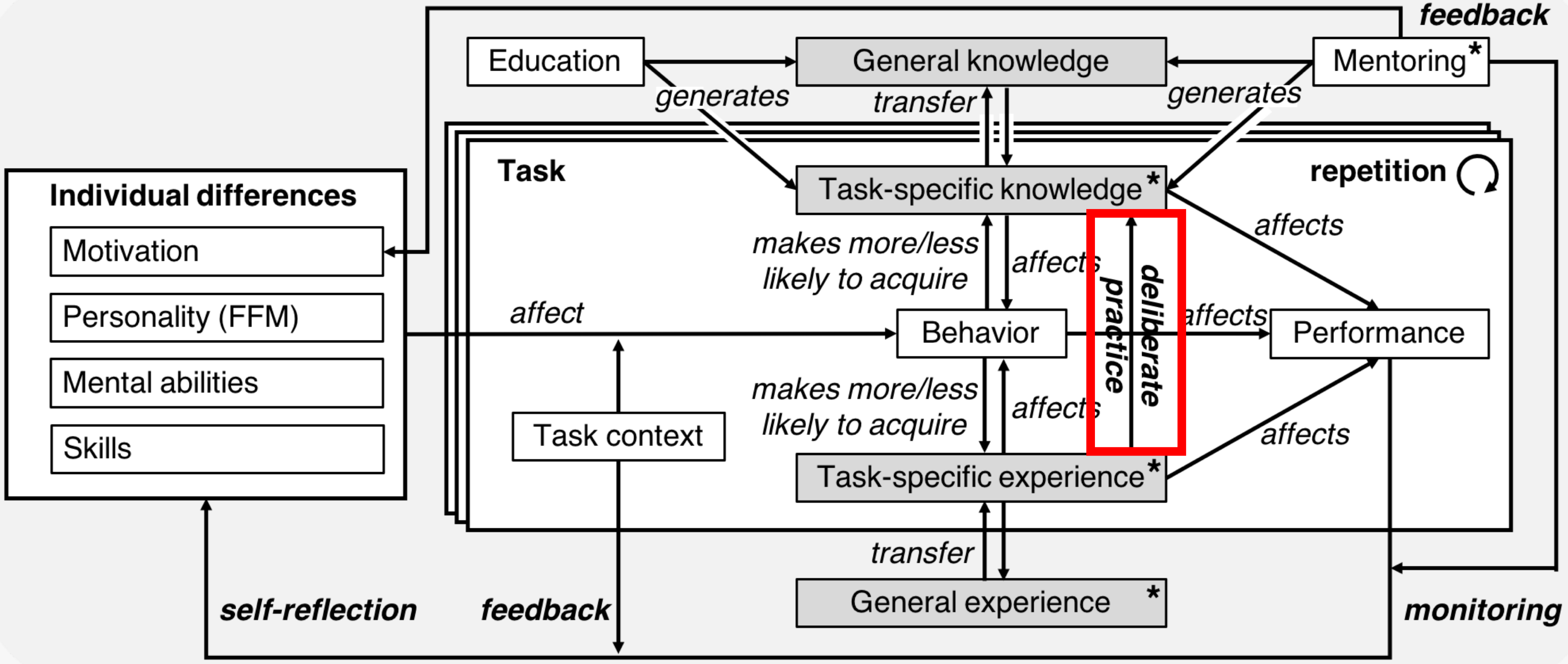
- Work **environment**
  (office, coworkers, customers etc.)
- Project **constraints**
  (external dependencies, time, etc.)
- Can either **foster or hinder** expertise dev.
- We asked: *What can employers do?*

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



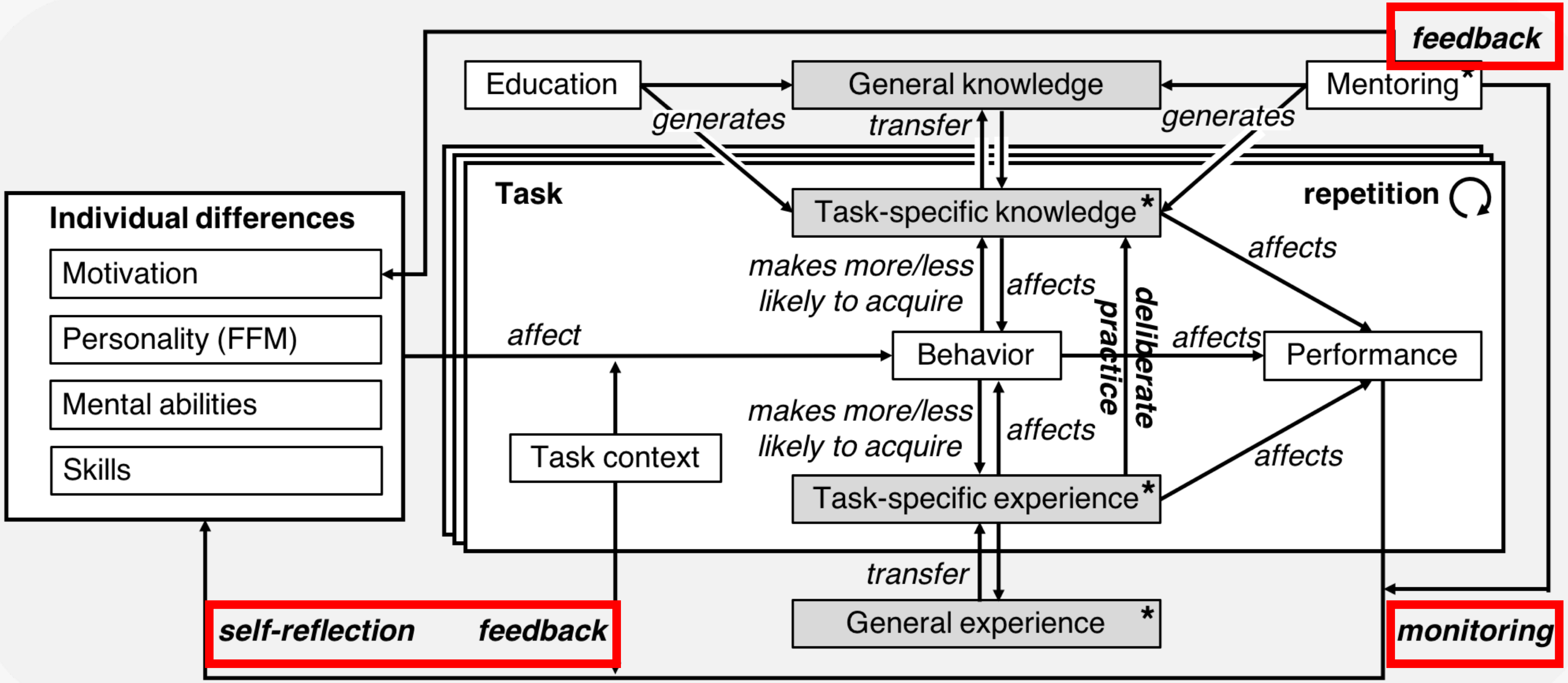
# Deliberate Practice

- 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)

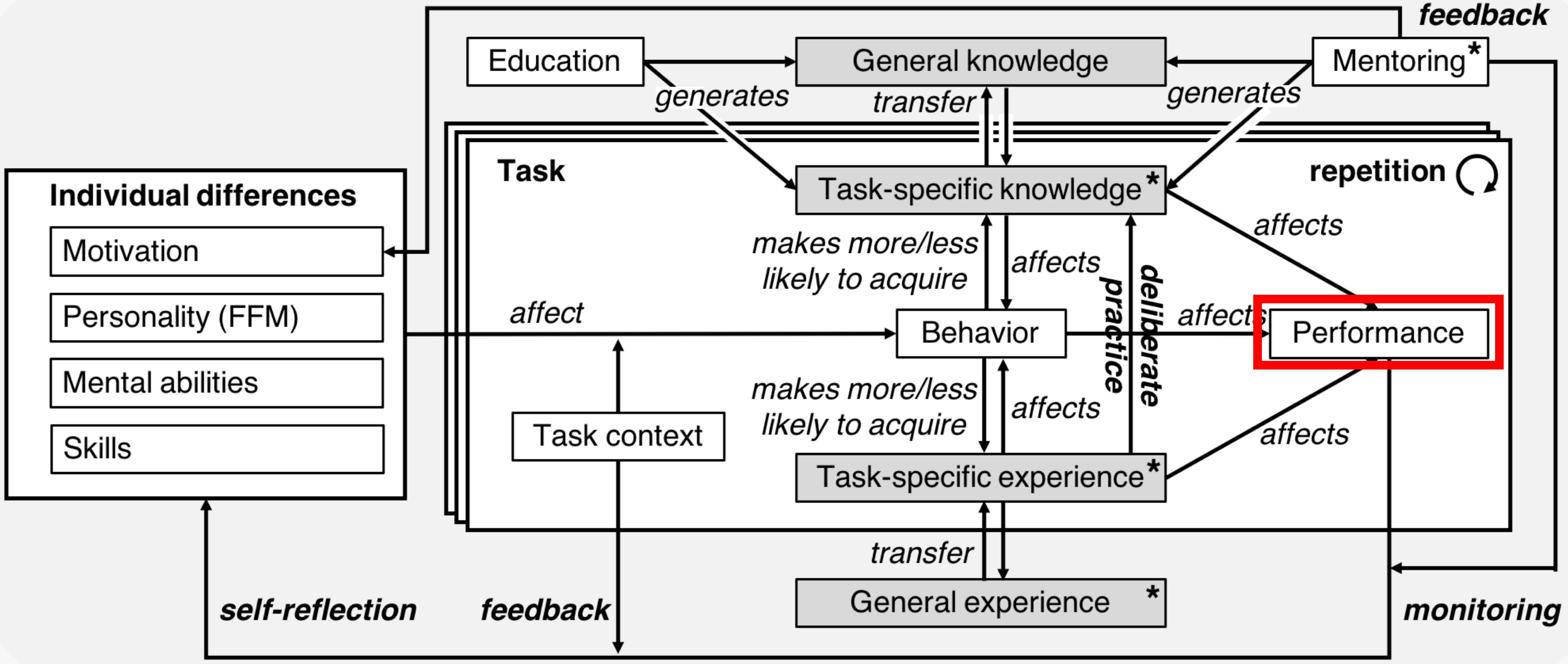
## Deliberate Practice: Self-Reflection

- **(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)
- **Mentors**, teachers, and peers are an important sources for feedback

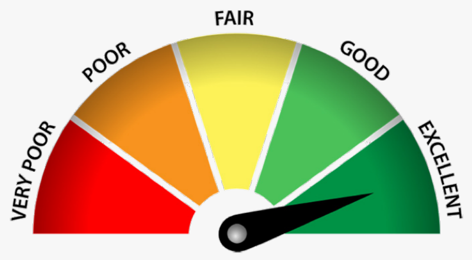
# Final Conceptual Theory



# Final Conceptual Theory



# Performance



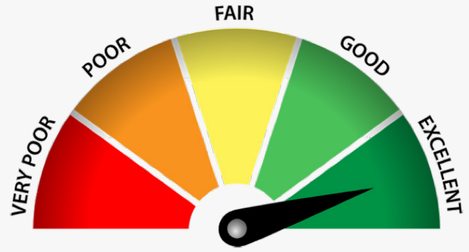
Scope of this work:

- 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



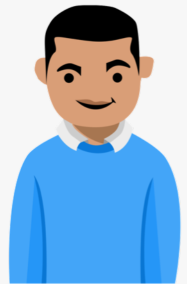
- Participants described different **properties of expert's source code** (well-structured, readable, maintainable, etc.)

*„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****."*



## 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**

## Semantic Differential Scale

- Beginning of survey:
  *Please rate your Java programming expertise on the following scale:*

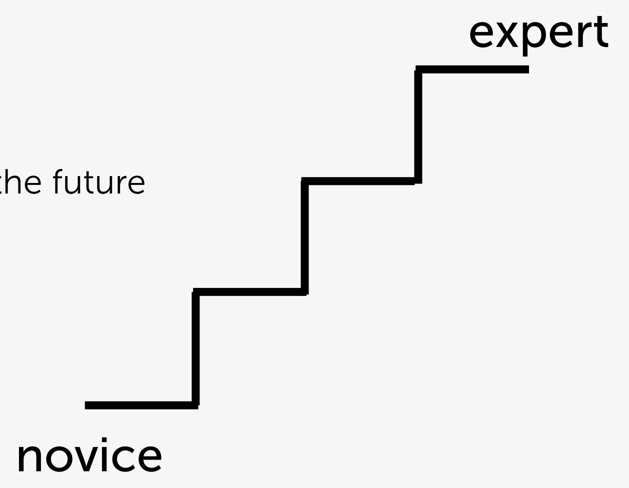
| | | | | | |
|---|---|---|---|---|---|
| *1 (Novice)* | *2* | *3* | *4* | *5* | *6 (Expert)* |
| ○ | ○ | ○ | ○ | ○ | ○ |

…

- End of survey:
  *Please rate your own Java programming expertise according to the five stages described below.*

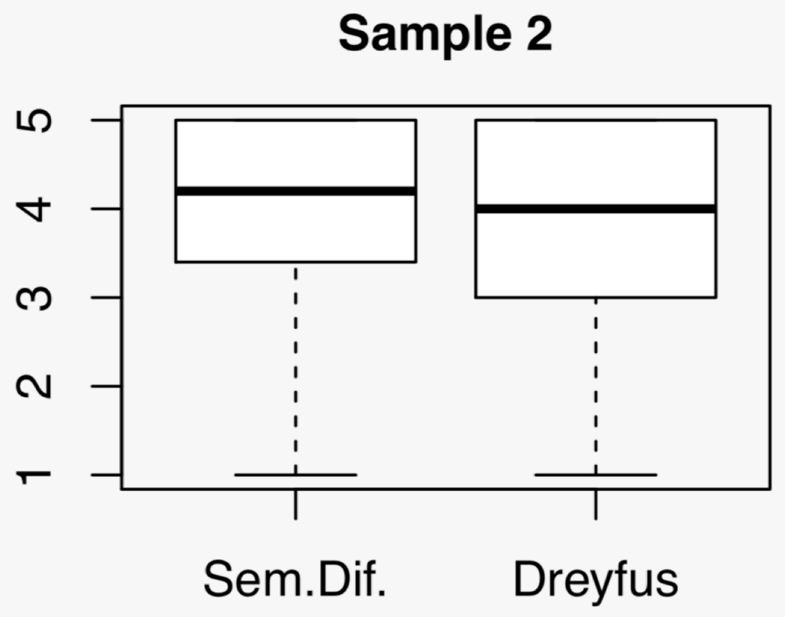
# Discrete Expertise Model

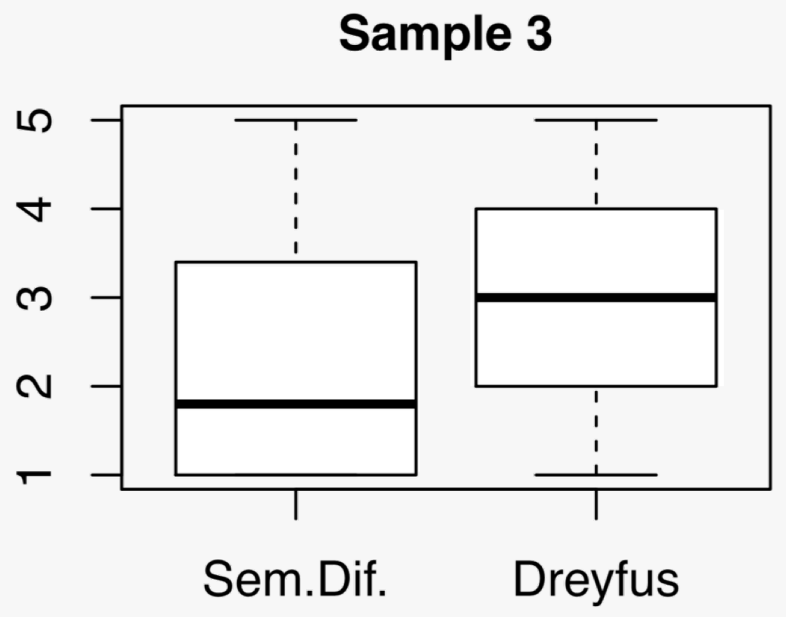
- Stage 1 (Novice):
  - has little or no experience
  - wants unambiguous rules to accomplish his/her tasks
  - is able to handle small, isolated tasks
- Stage 2 (Advanced Beginner):
  - has gained some experience
  - can work more independently than a novice
  - knows general principles in a limited 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





## Takeaways

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

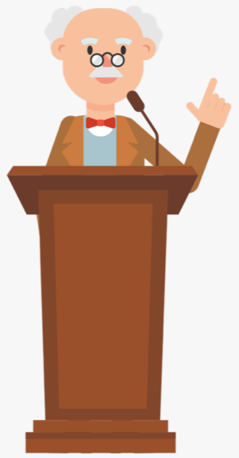
# Summary for Researchers

- 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

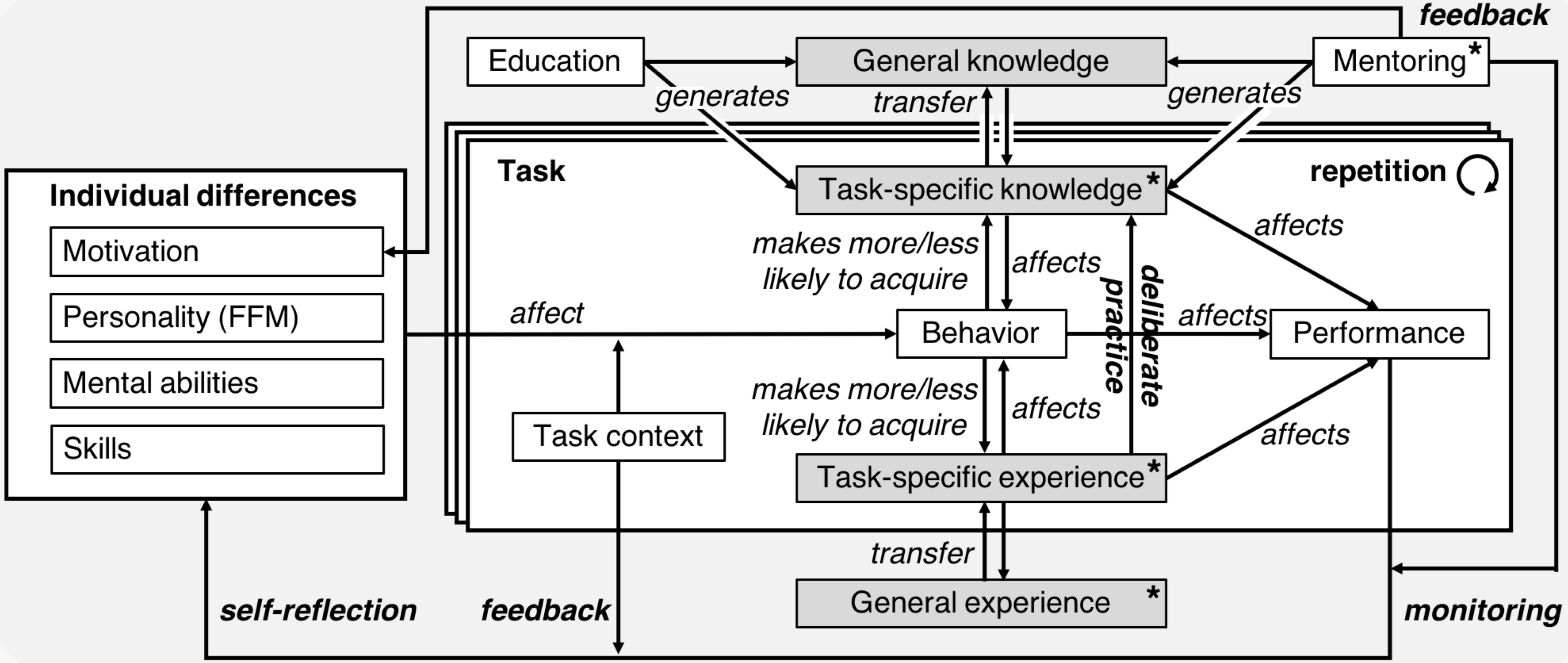
- See which **attributes** other developers assign to experts
- Learn which **behaviors** may lead to becoming a better software developer:
  - Deliberate practice
  - Have challenging goals
  - Build or maintain a supportive work environment (also for others)
  - Ask for feedback from peers
  - Reflect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

# Summary for Employers

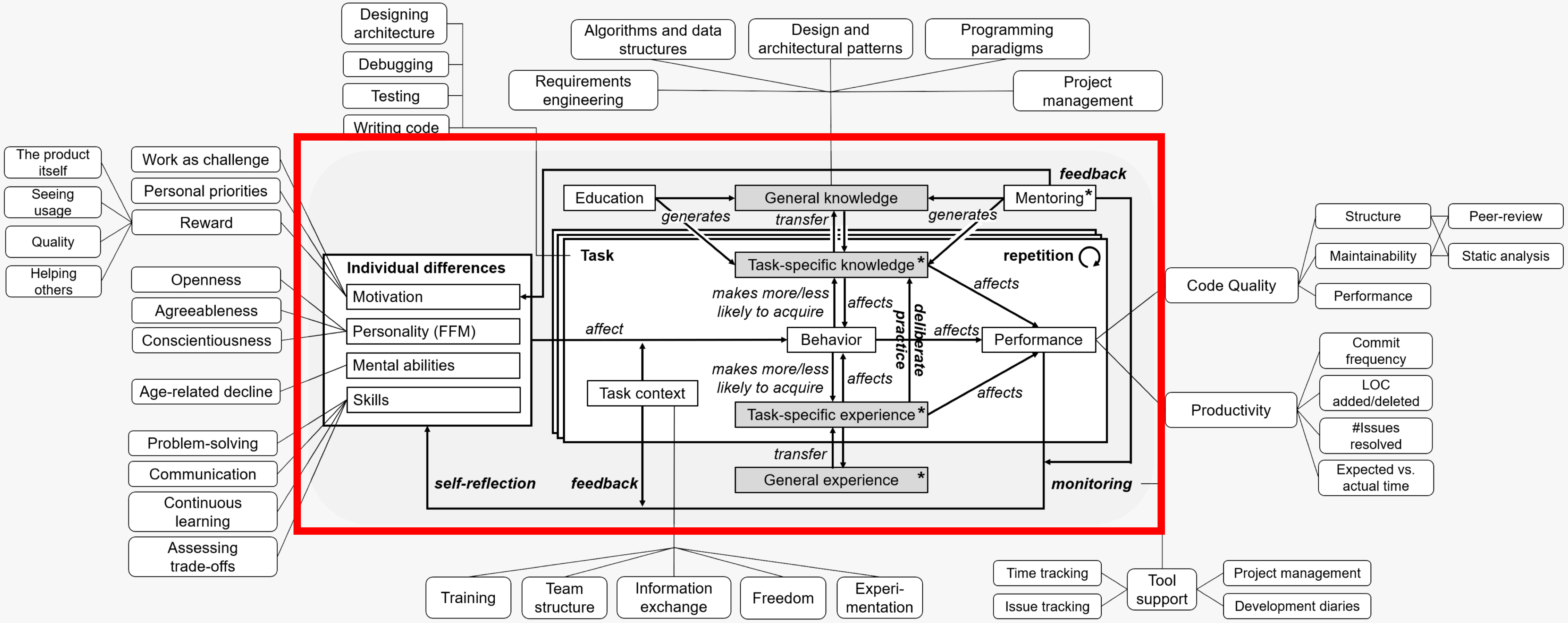
- Learn what **(de)motivates** their employees:
  - Main motivation: problem solving
  - Main demotivation: non-challenging work

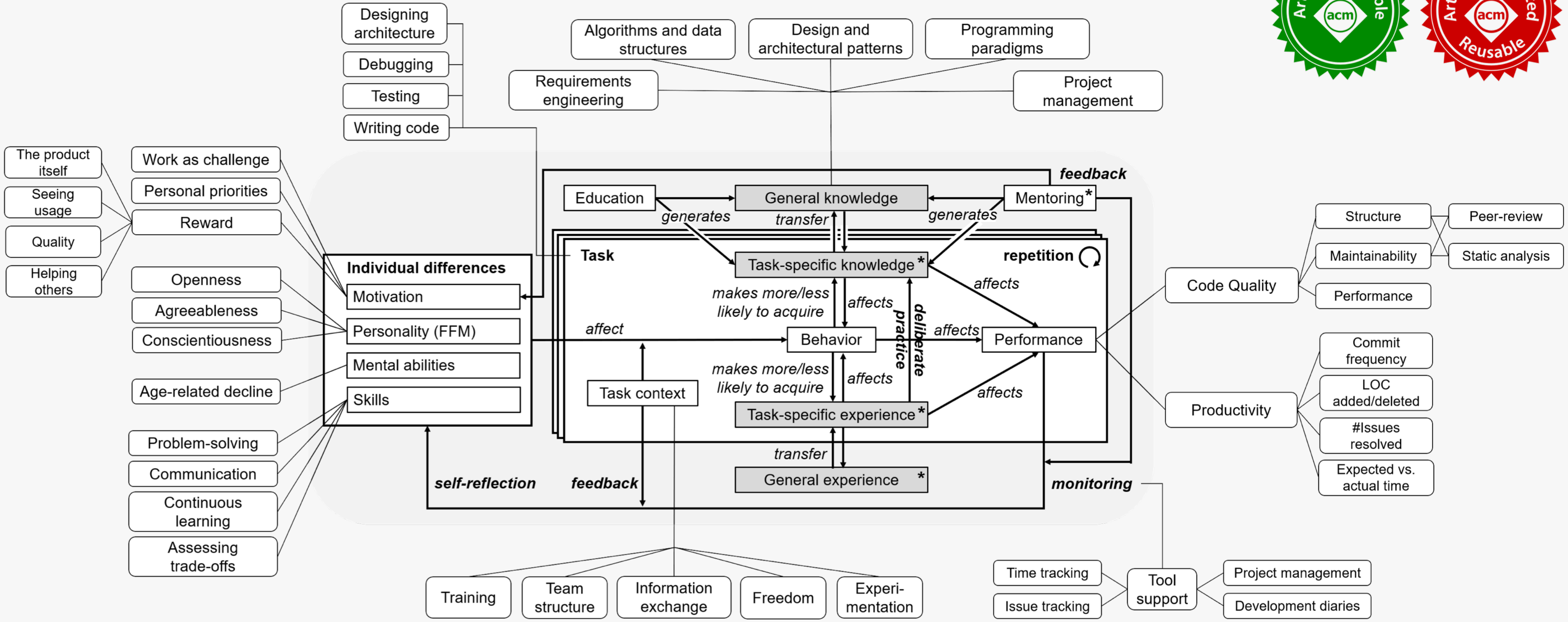
- Ideas on how to build supportive work environment **supporting self-improvement** of staff:
  - Good mix of continuity and change in software development process
  - Communicate clear visions, directions, and goals
  - Reward high-quality work wherever possible
  - Revisit information sharing in company
  - Facilitate meetings

# Core of Conceptual Theory



# Complete Conceptual Theory





**Sebastian Baltes**

@s_baltes

# expertise.sbaltes.com

*Data and scripts available on Zenodo*

## Interested in pursuing a PhD in Australia?

**Open topic**

Broad area of (empirical) software engineering

**Fully funded**

Scholarship includes tuition fees & living expenses

**Application deadline:**

20 December 2019

Sebastian Baltes
@s_baltes

empirical-software.engineering