Software Developers‘ Work Habits and Expertise

Sketching, Code Plagiarism, and Expertise Development

Sebastian Baltes

@s_baltes

Universität Trier

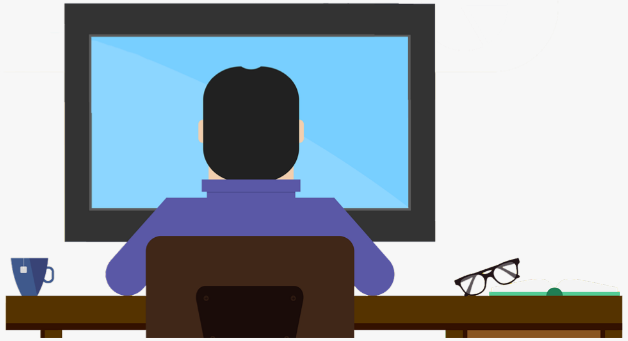
empirical-software.engineering

Habit

Merriam-Webster

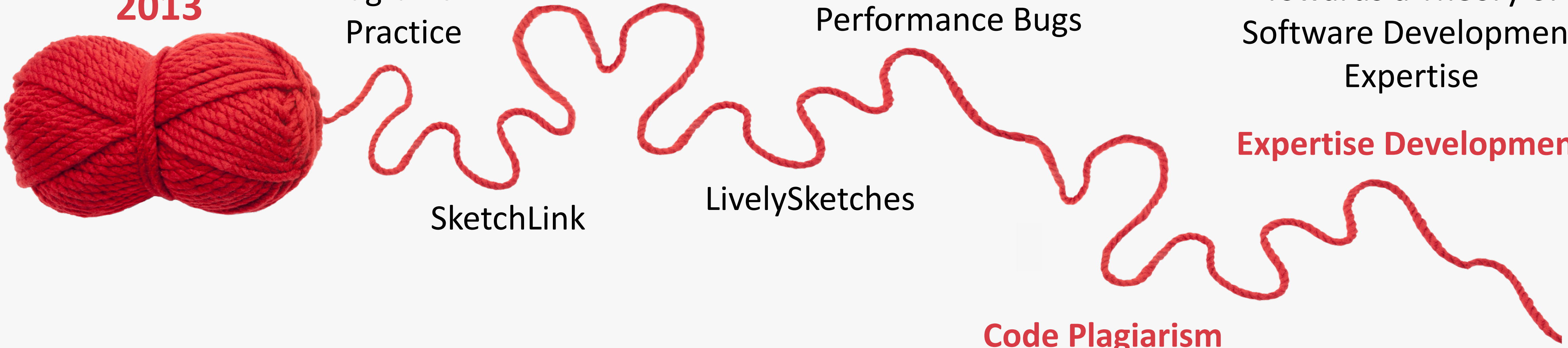
„a **settled tendency** or **usual manner of behavior**“

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

LivelySketches

How Developers Locate Performance Bugs

Code Plagiarism

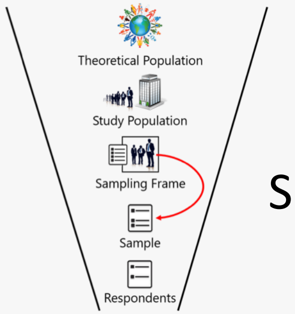
stackoverflow Code Snippets in GitHub Projects

Towards a Theory of Software Development Expertise

Expertise Development

2018

“Parallel Thread“



Issues in Sampling Software Developers

2013

Methodology

Open Data

SOTorrent

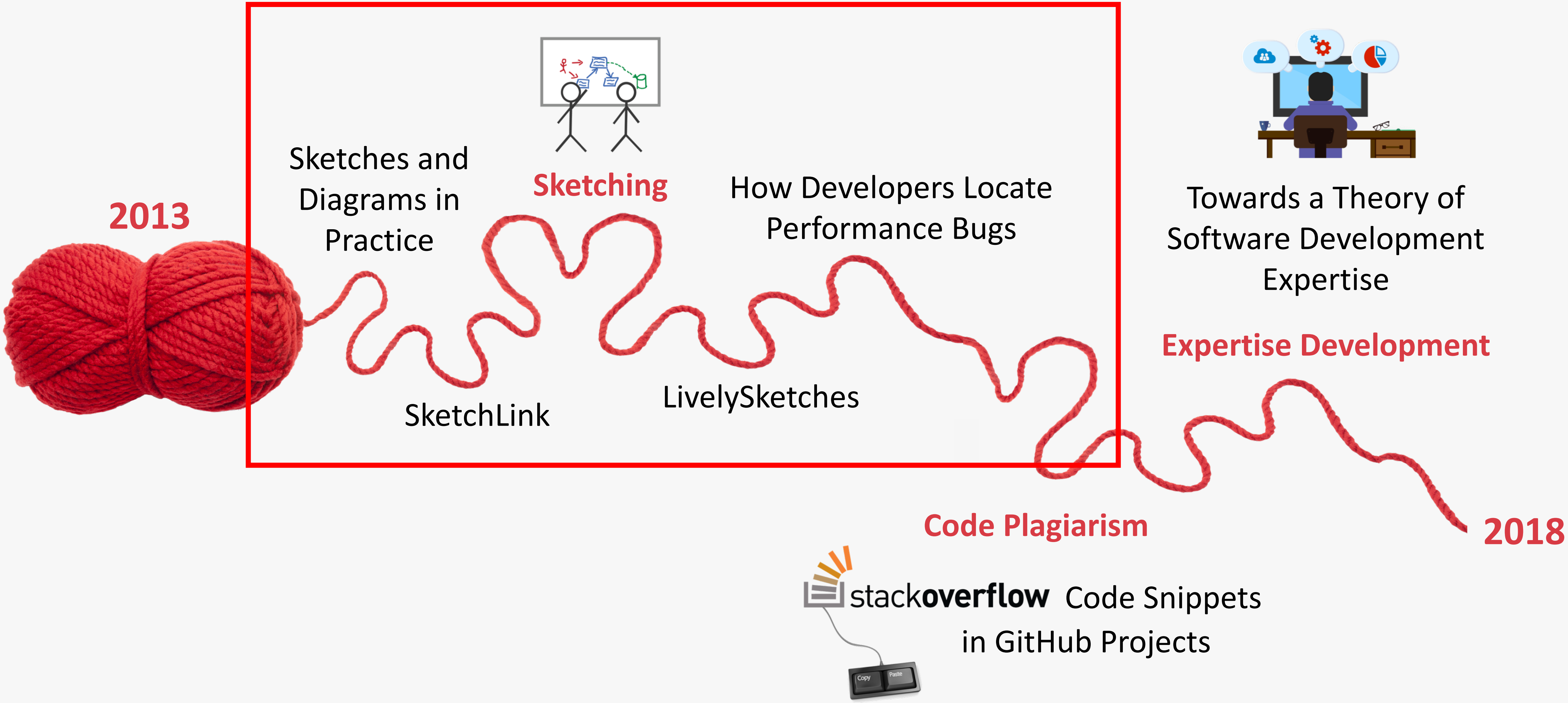
airbnb

Constructing Urban Tourism Space Digitally

Interdisciplinary Research

2018

Studied Habits



2013

Sketches and Diagrams in Practice

Sketching

How Developers Locate Performance Bugs

SketchLink

LivelySketches

Towards a Theory of Software Development Expertise

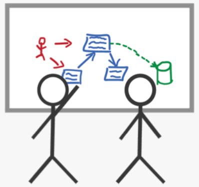
Expertise Development

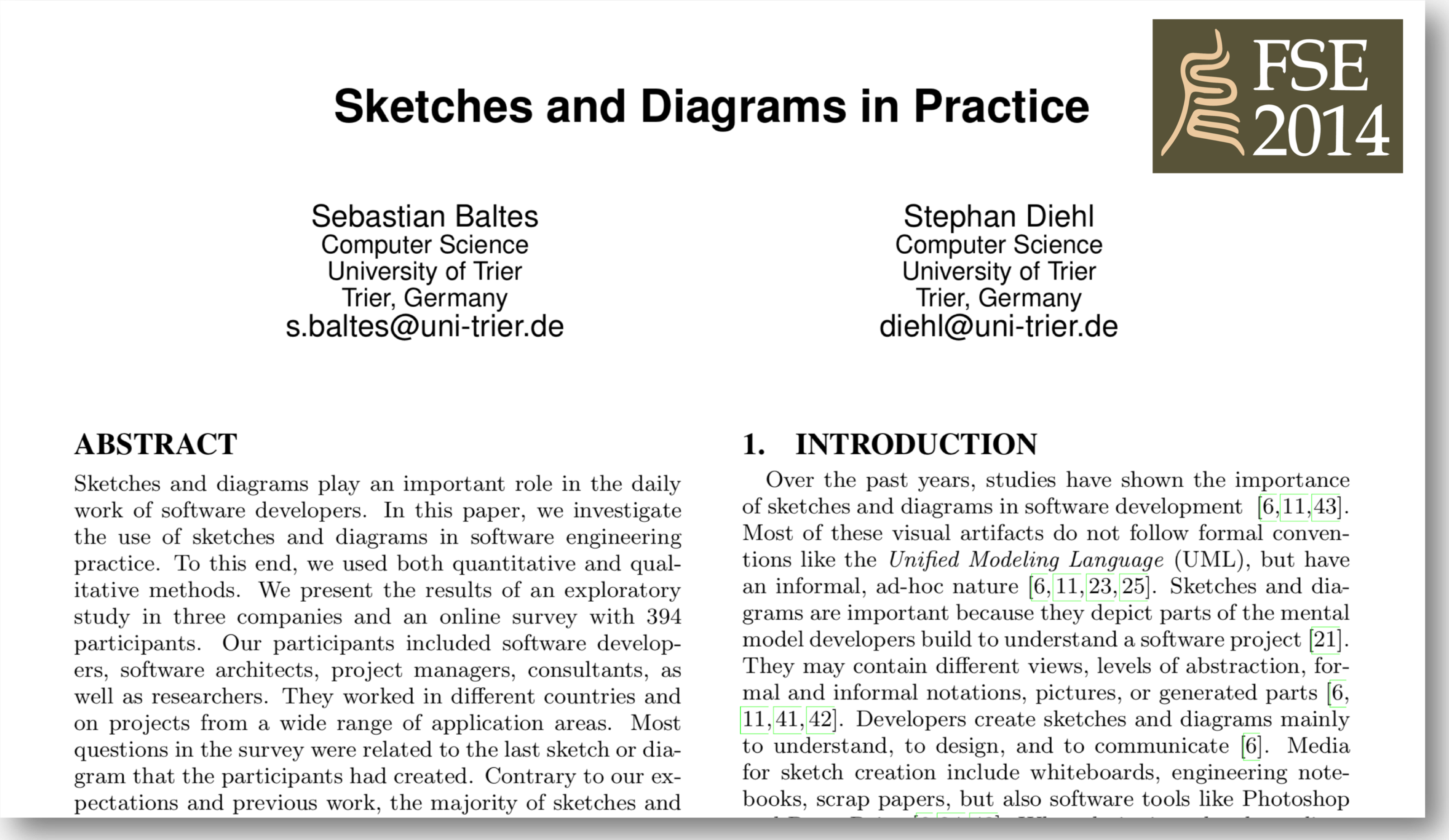
Code Plagiarism

stackoverflow Code Snippets in GitHub Projects

2018

Sketching





FSE 2014

Sketches and Diagrams in Practice

Sebastian Baltes
Computer Science
University of Trier
Trier, Germany
s.baltes@uni-trier.de

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

ABSTRACT

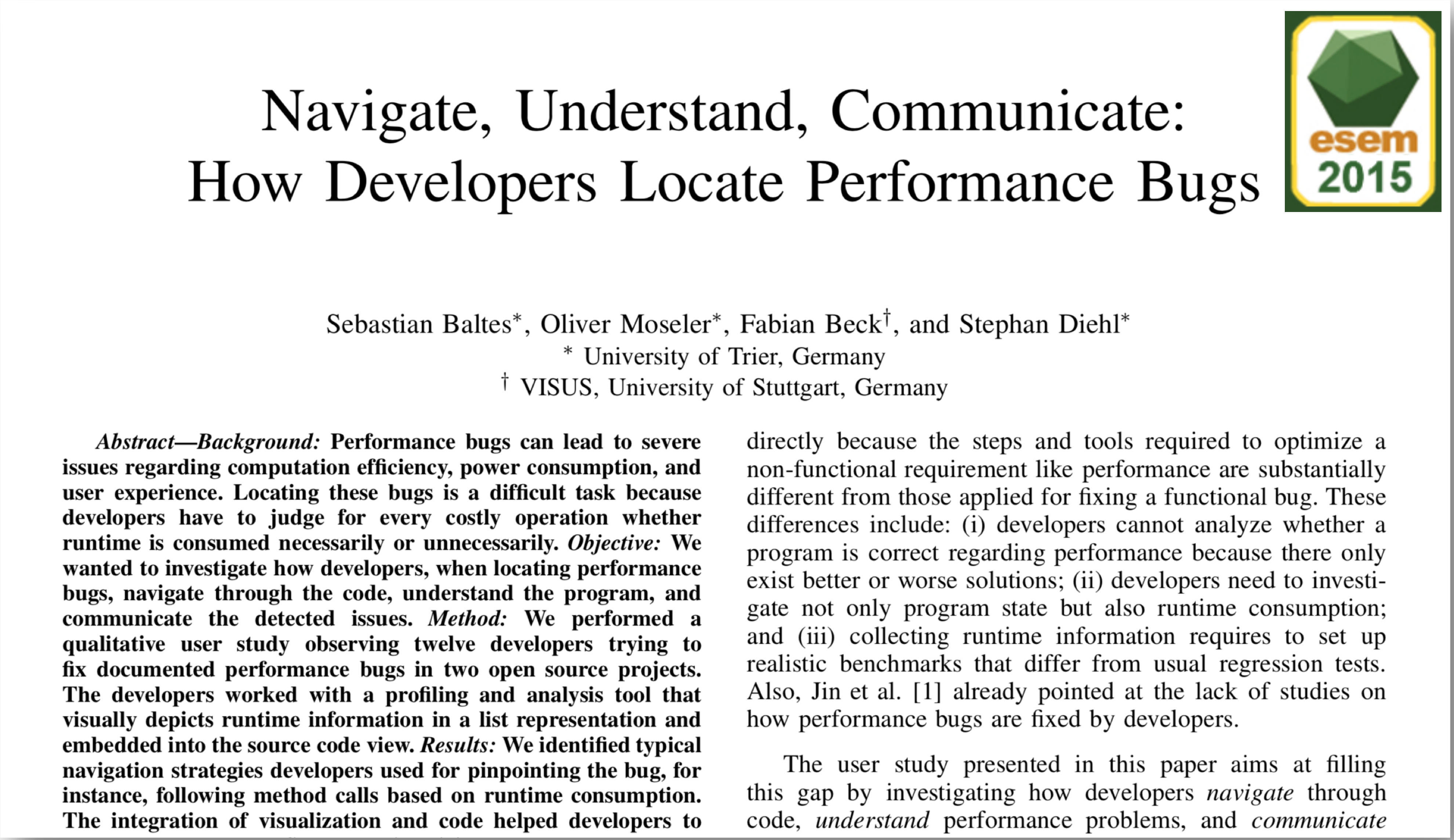
Sketches and diagrams play an important role in the daily work of software developers. In this paper, we investigate the use of sketches and diagrams in software engineering practice. To this end, we used both quantitative and qualitative methods. We present the results of an exploratory study in three companies and an online survey with 394 participants. Our participants included software developers, software architects, project managers, consultants, as well as researchers. They worked in different countries and on projects from a wide range of application areas. Most questions in the survey were related to the last sketch or diagram that the participants had created. Contrary to our expectations and previous work, the majority of sketches and

1. INTRODUCTION

Over the past years, studies have shown the importance of sketches and diagrams in software development [6,11,43]. Most of these visual artifacts do not follow formal conventions like the *Unified Modeling Language* (UML), but have an informal, ad-hoc nature [6,11,23,25]. Sketches and diagrams are important because they depict parts of the mental model developers build to understand a software project [21]. They may contain different views, levels of abstraction, formal and informal notations, pictures, or generated parts [6, 11,41,42]. Developers create sketches and diagrams mainly to understand, to design, and to communicate [6]. Media for sketch creation include whiteboards, engineering notebooks, scrap papers, but also software tools like Photoshop

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

Sketching



Navigate, Understand, Communicate: How Developers Locate Performance Bugs

esem 2015

Sebastian Baltes*, Oliver Moseler*, Fabian Beck†, and Stephan Diehl*
* University of Trier, Germany
† VISUS, University of Stuttgart, Germany

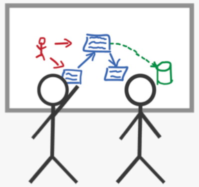
***Abstract—Background:* Performance bugs can lead to severe issues regarding computation efficiency, power consumption, and user experience. Locating these bugs is a difficult task because developers have to judge for every costly operation whether runtime is consumed necessarily or unnecessarily. *Objective:* We wanted to investigate how developers, when locating performance bugs, navigate through the code, understand the program, and communicate the detected issues. *Method:* We performed a qualitative user study observing twelve developers trying to fix documented performance bugs in two open source projects. The developers worked with a profiling and analysis tool that visually depicts runtime information in a list representation and embedded into the source code view. *Results:* We identified typical navigation strategies developers used for pinpointing the bug, for instance, following method calls based on runtime consumption. The integration of visualization and code helped developers to**

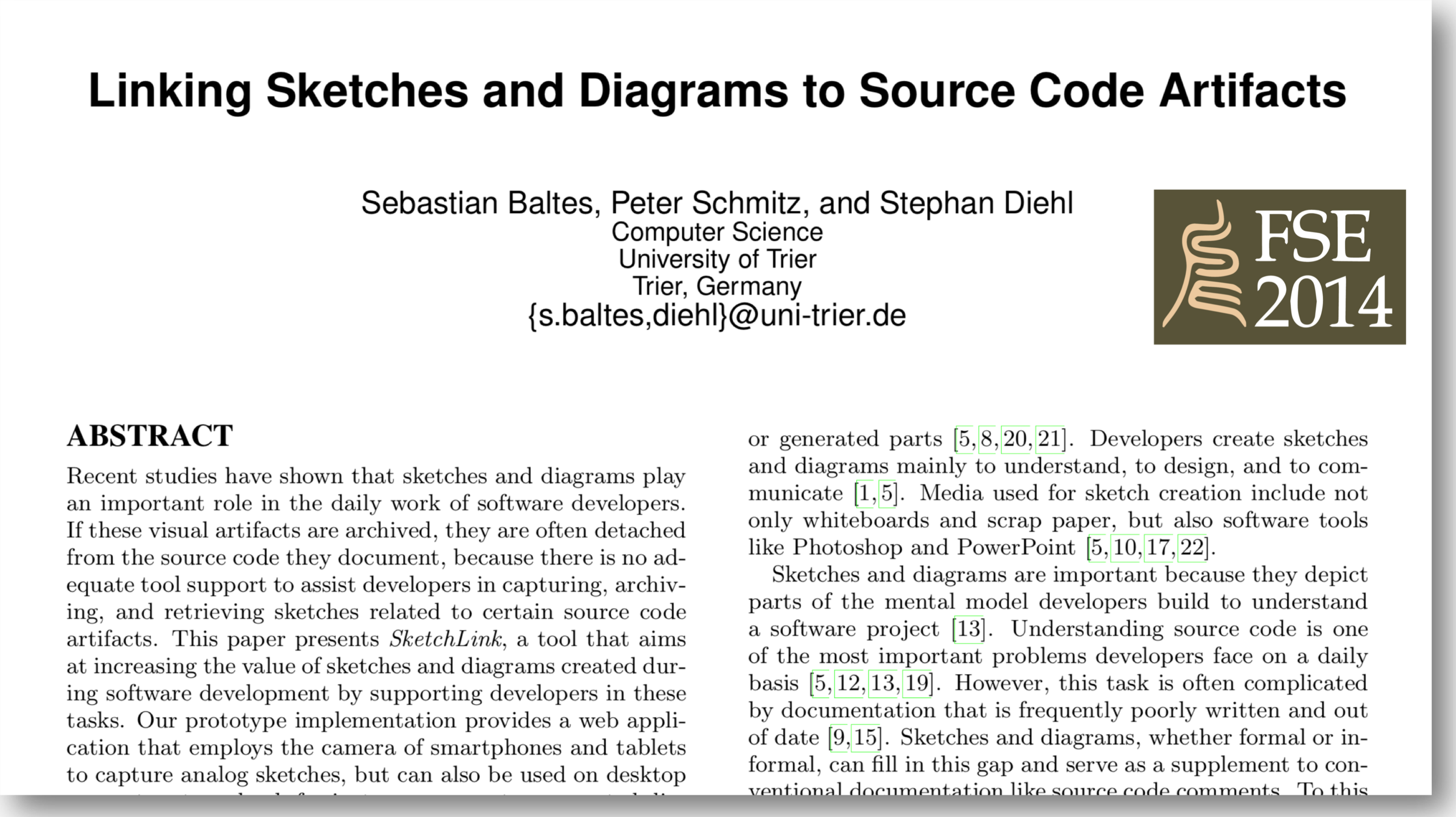
directly because the steps and tools required to optimize a non-functional requirement like performance are substantially different from those applied for fixing a functional bug. These differences include: (i) developers cannot analyze whether a program is correct regarding performance because there only exist better or worse solutions; (ii) developers need to investigate not only program state but also runtime consumption; and (iii) collecting runtime information requires to set up realistic benchmarks that differ from usual regression tests. Also, Jin et al. [1] already pointed at the lack of studies on how performance bugs are fixed by developers.

The user study presented in this paper aims at filling this gap by investigating how developers *navigate* through code, *understand* performance problems, and *communicate*

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

Sketching





Linking Sketches and Diagrams to Source Code Artifacts

Sebastian Baltes, Peter Schmitz, and Stephan Diehl
Computer Science
University of Trier
Trier, Germany
{s.baltes,diehl}@uni-trier.de

FSE 2014

ABSTRACT

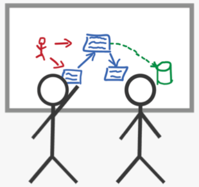
Recent studies have shown that sketches and diagrams play an important role in the daily work of software developers. If these visual artifacts are archived, they are often detached from the source code they document, because there is no adequate tool support to assist developers in capturing, archiving, and retrieving sketches related to certain source code artifacts. This paper presents *SketchLink*, a tool that aims at increasing the value of sketches and diagrams created during software development by supporting developers in these tasks. Our prototype implementation provides a web application that employs the camera of smartphones and tablets to capture analog sketches, but can also be used on desktop

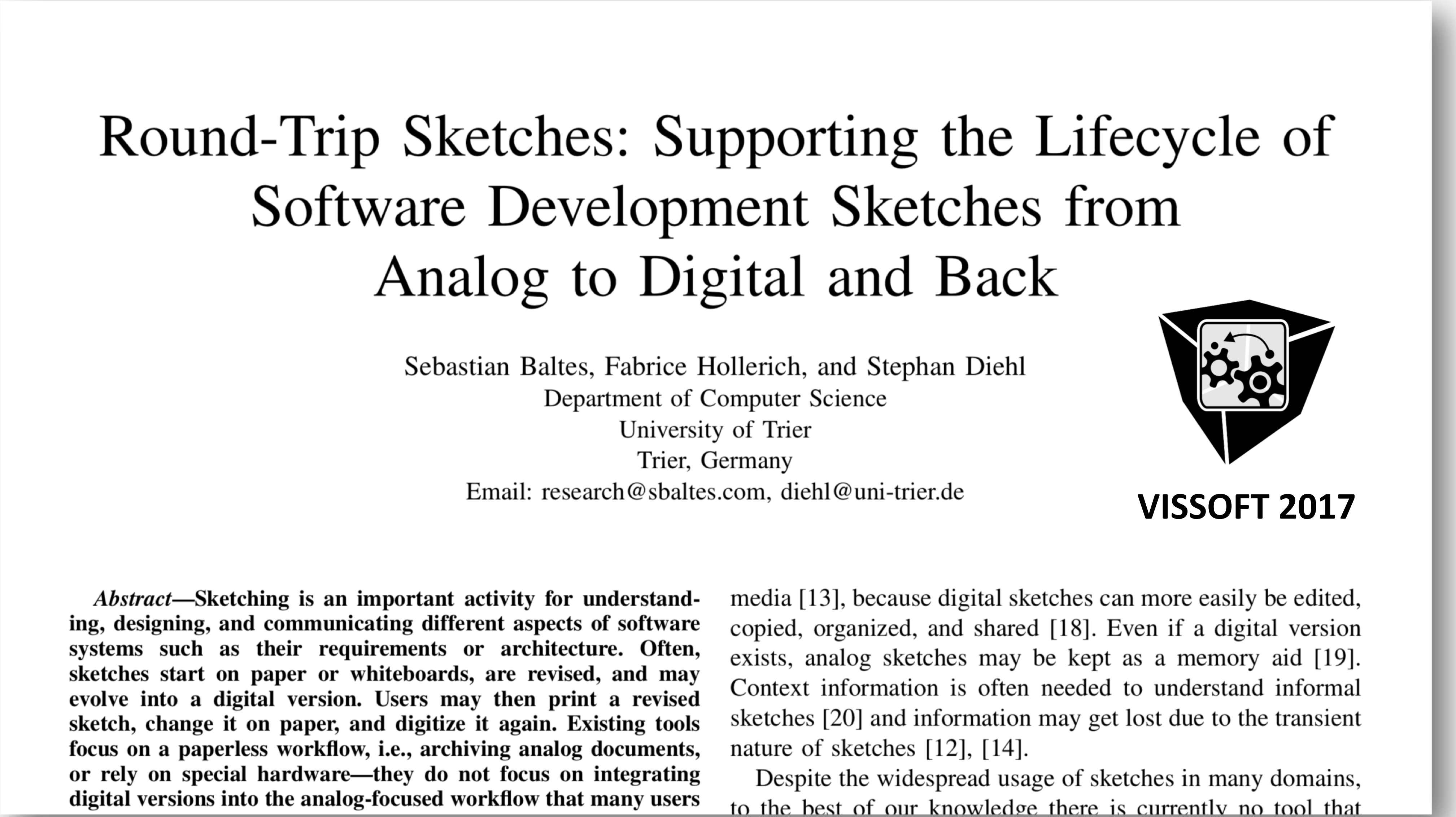
or generated parts [5,8,20,21]. Developers create sketches and diagrams mainly to understand, to design, and to communicate [1,5]. Media used for sketch creation include not only whiteboards and scrap paper, but also software tools like Photoshop and PowerPoint [5,10,17,22].

Sketches and diagrams are important because they depict parts of the mental model developers build to understand a software project [13]. Understanding source code is one of the most important problems developers face on a daily basis [5,12,13,19]. However, this task is often complicated by documentation that is frequently poorly written and out of date [9,15]. Sketches and diagrams, whether formal or informal, can fill in this gap and serve as a supplement to conventional documentation like source code comments. To this

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

Sketching





Round-Trip Sketches: Supporting the Lifecycle of Software Development Sketches from Analog to Digital and Back

Sebastian Baltes, Fabrice Hollerich, and Stephan Diehl
Department of Computer Science
University of Trier
Trier, Germany
Email: research@sbaltes.com, diehl@uni-trier.de

***Abstract*—Sketching is an important activity for understanding, designing, and communicating different aspects of software systems such as their requirements or architecture. Often, sketches start on paper or whiteboards, are revised, and may evolve into a digital version. Users may then print a revised sketch, change it on paper, and digitize it again. Existing tools focus on a paperless workflow, i.e., archiving analog documents, or rely on special hardware—they do not focus on integrating digital versions into the analog-focused workflow that many users**

media [13], because digital sketches can more easily be edited, copied, organized, and shared [18]. Even if a digital version exists, analog sketches may be kept as a memory aid [19]. Context information is often needed to understand informal sketches [20] and information may get lost due to the transient nature of sketches [12], [14].

Despite the widespread usage of sketches in many domains, to the best of our knowledge there is currently no tool that

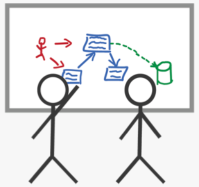
VISSOFT 2017

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

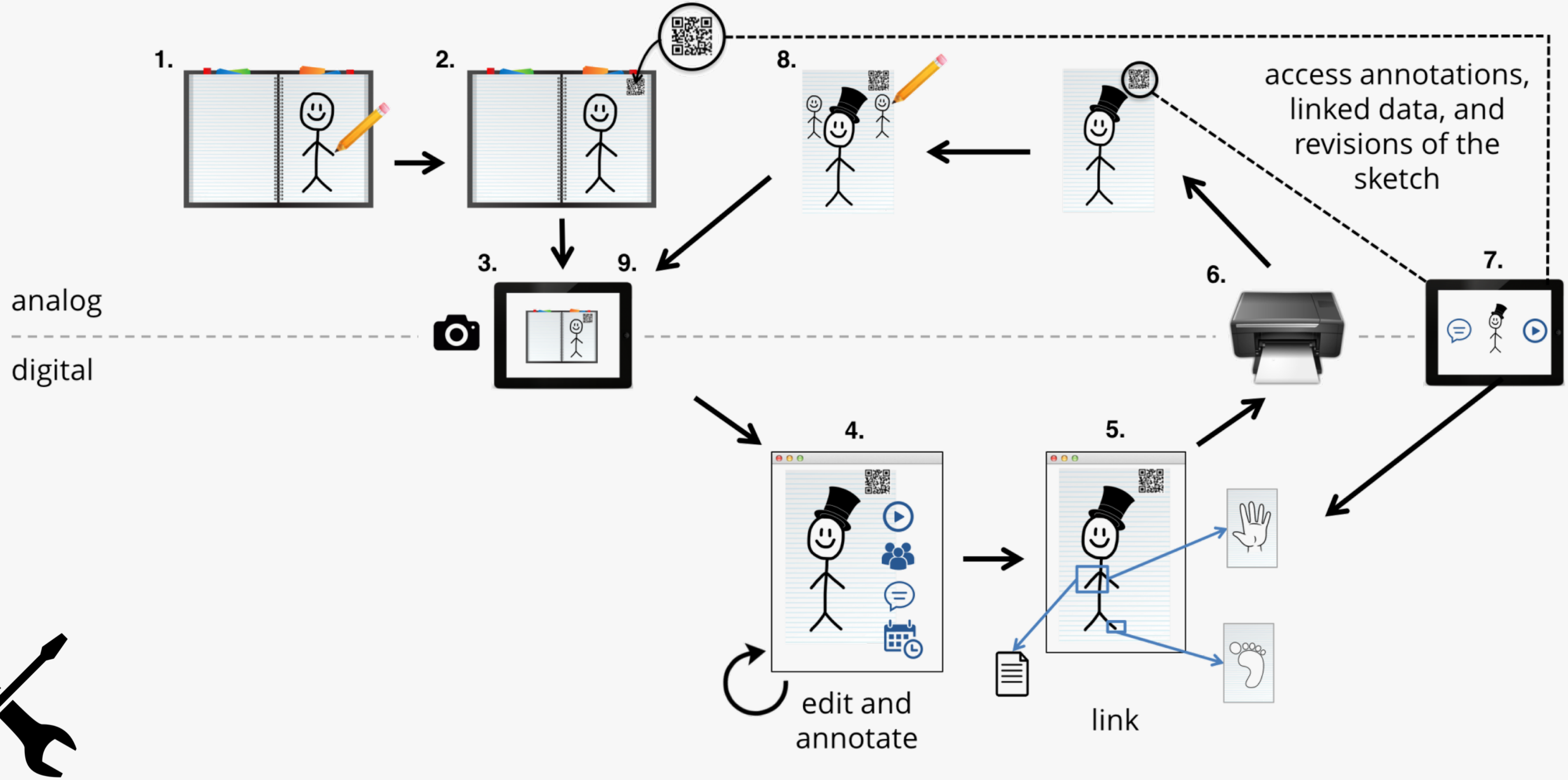
Sketching

SketchLink

https://www.youtube.com/watch?v=mG6xCiQpS80



LivelySketches



access annotations, linked data, and revisions of the sketch

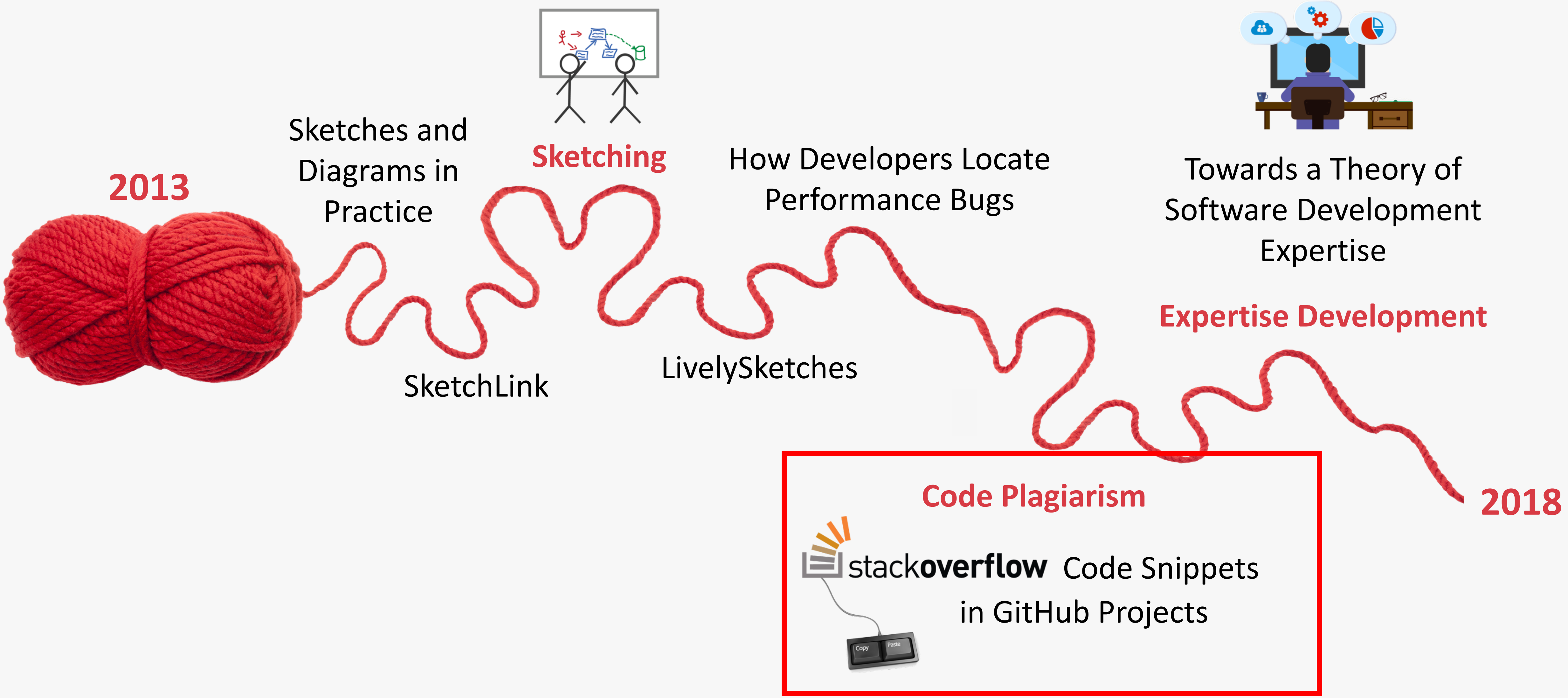
analog

digital

edit and annotate

link

Studied Habits



2013

Sketches and Diagrams in Practice

Sketching

SketchLink

LivelySketches

How Developers Locate Performance Bugs

Towards a Theory of Software Development Expertise

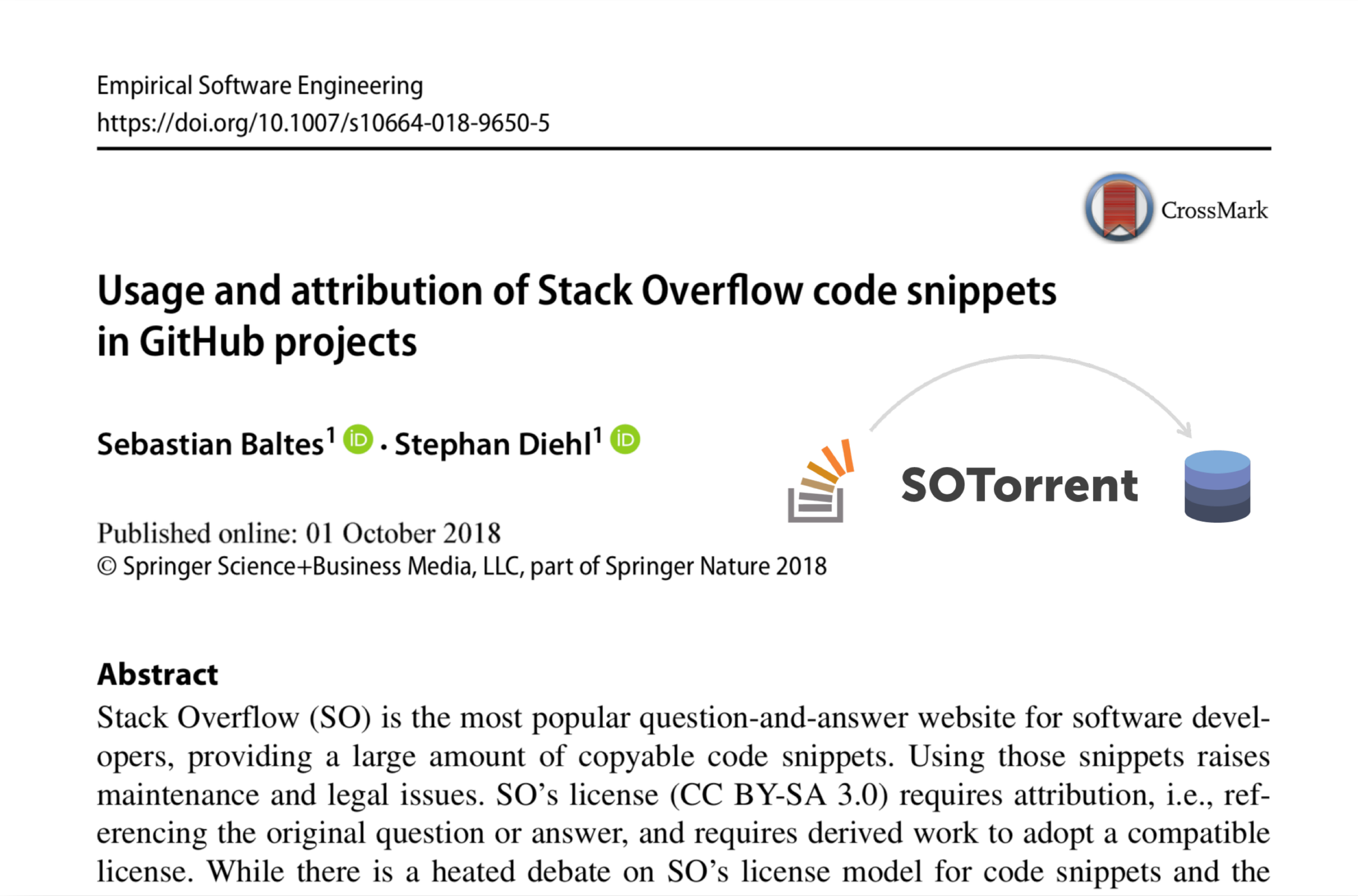
Expertise Development

Code Plagiarism

stackoverflow Code Snippets in GitHub Projects

2018

Code Plagiarism



Empirical Software Engineering
https://doi.org/10.1007/s10664-018-9650-5

Usage and attribution of Stack Overflow code snippets in GitHub projects

Sebastian Baltes[1] · **Stephan Diehl**[1]

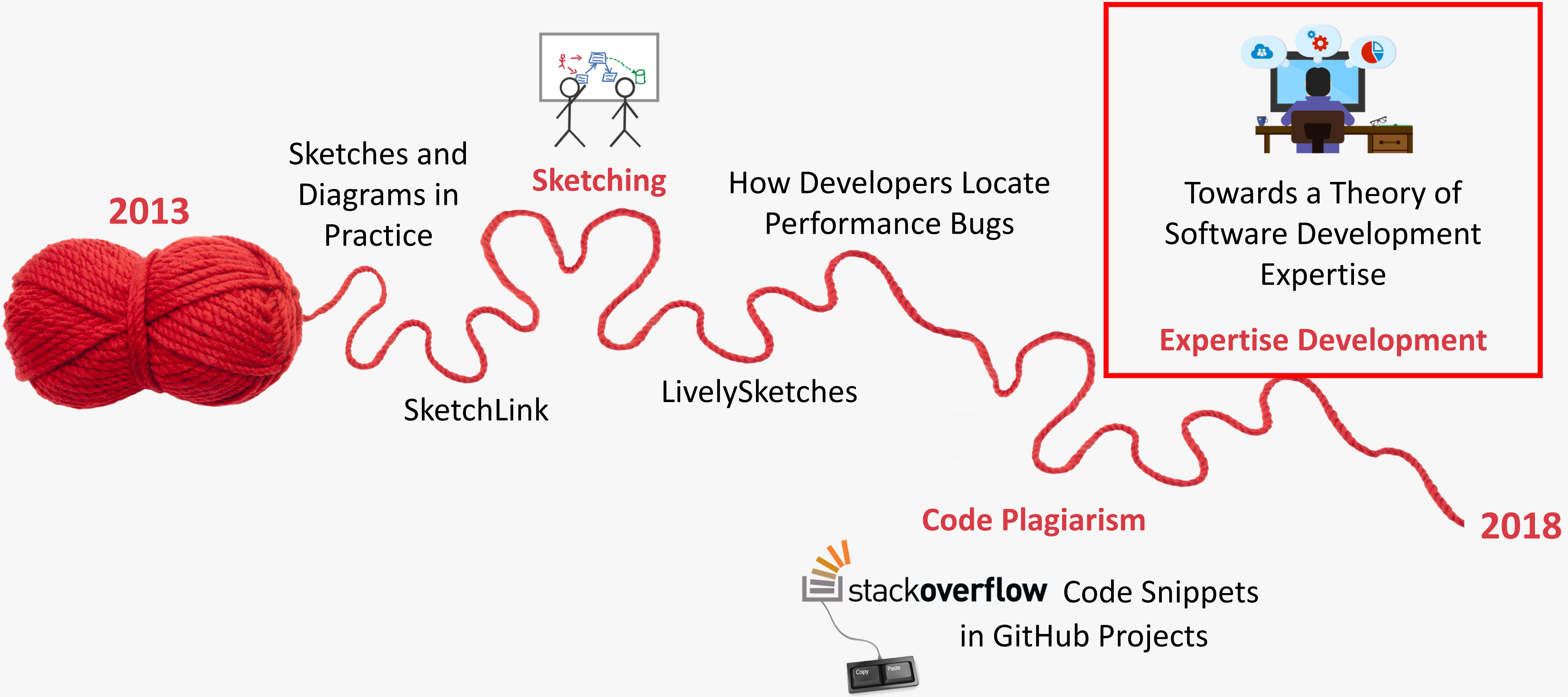
Published online: 01 October 2018

Abstract

Stack Overflow (SO) is the most popular question-and-answer website for software developers, providing a large amount of copyable code snippets. Using those snippets raises maintenance and legal issues. SO's license (CC BY-SA 3.0) requires attribution, i.e., referencing the original question or answer, and requires derived work to adopt a compatible license. While there is a heated debate on SO's license model for code snippets and the

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

Studied Habits



2013

Sketches and Diagrams in Practice

Sketching

SketchLink

LivelySketches

How Developers Locate Performance Bugs

Code Plagiarism

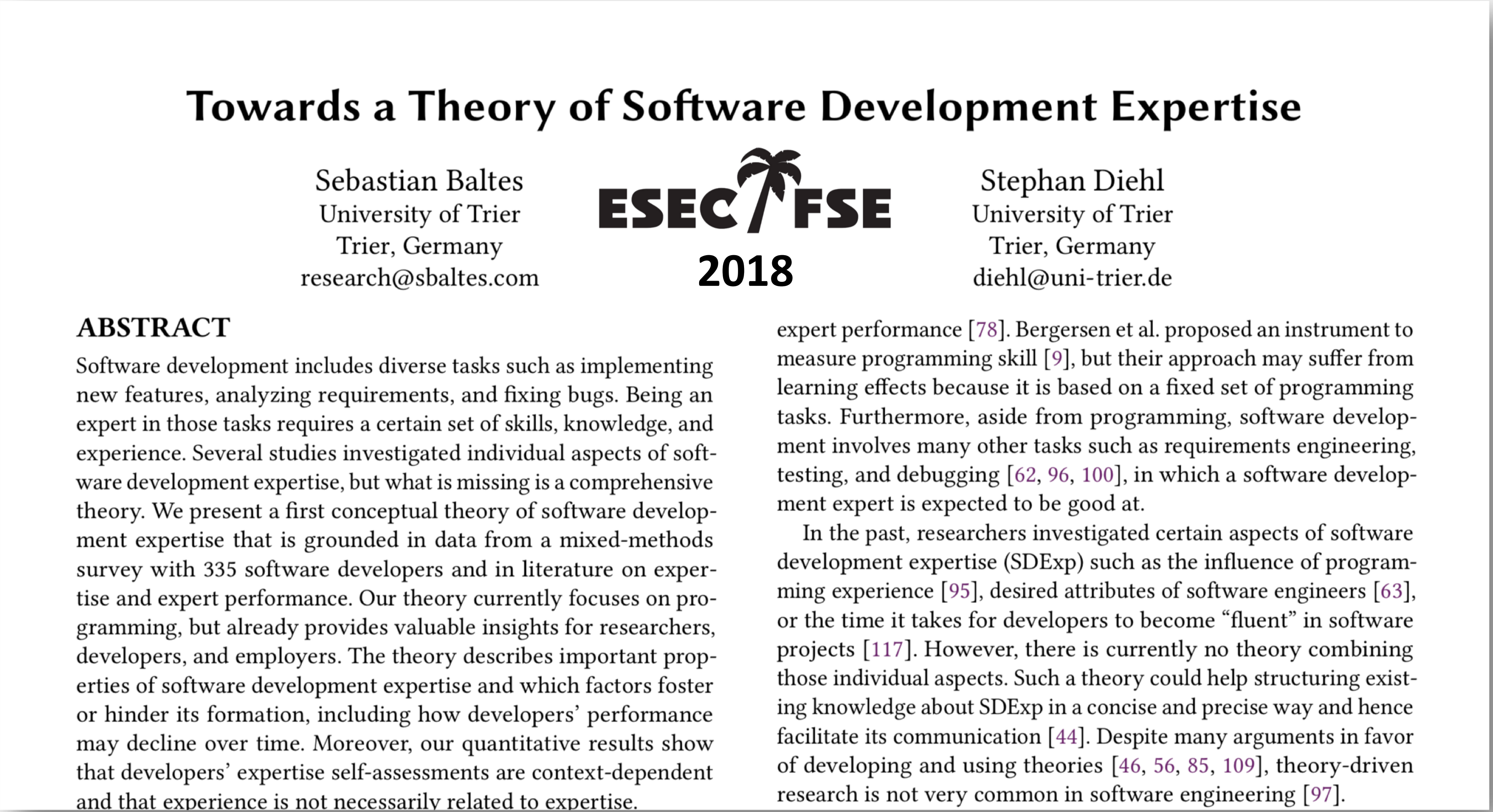
stackoverflow Code Snippets in GitHub Projects

Towards a Theory of Software Development Expertise

Expertise Development

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?

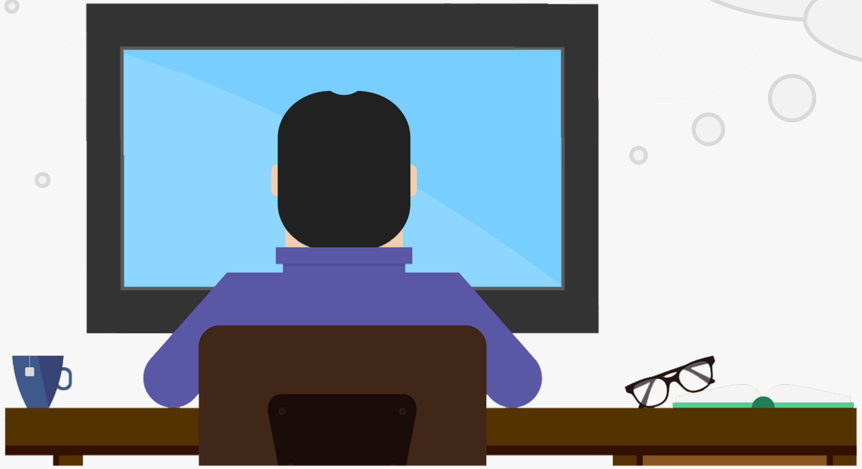
Implementing new features

Data structures

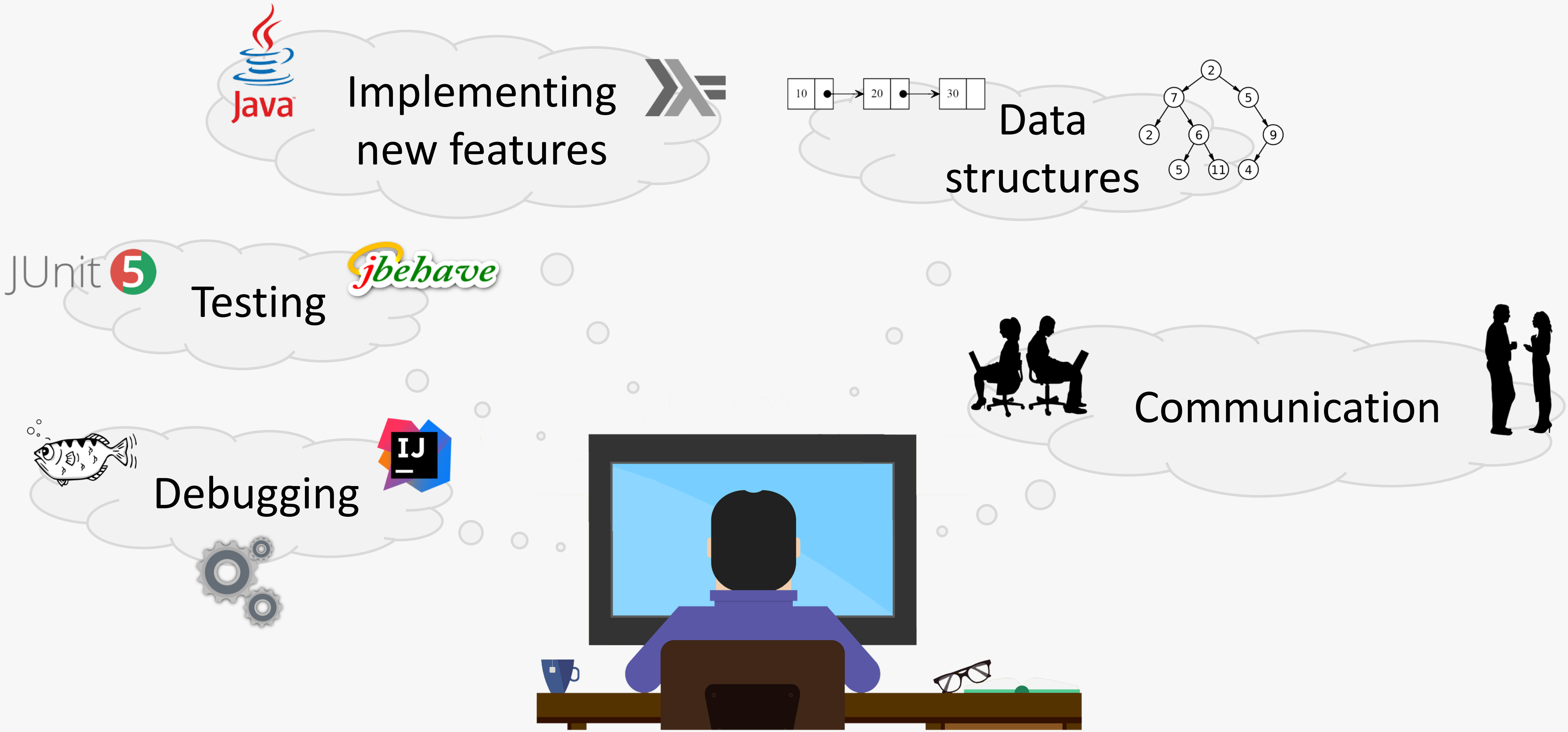
Testing

Communication

Debugging



Software Development Expertise?



Implementing new features

Data structures

Testing

Debugging

Communication

How to structure all those expertise-related aspects?

Which factors influence expertise development over time?

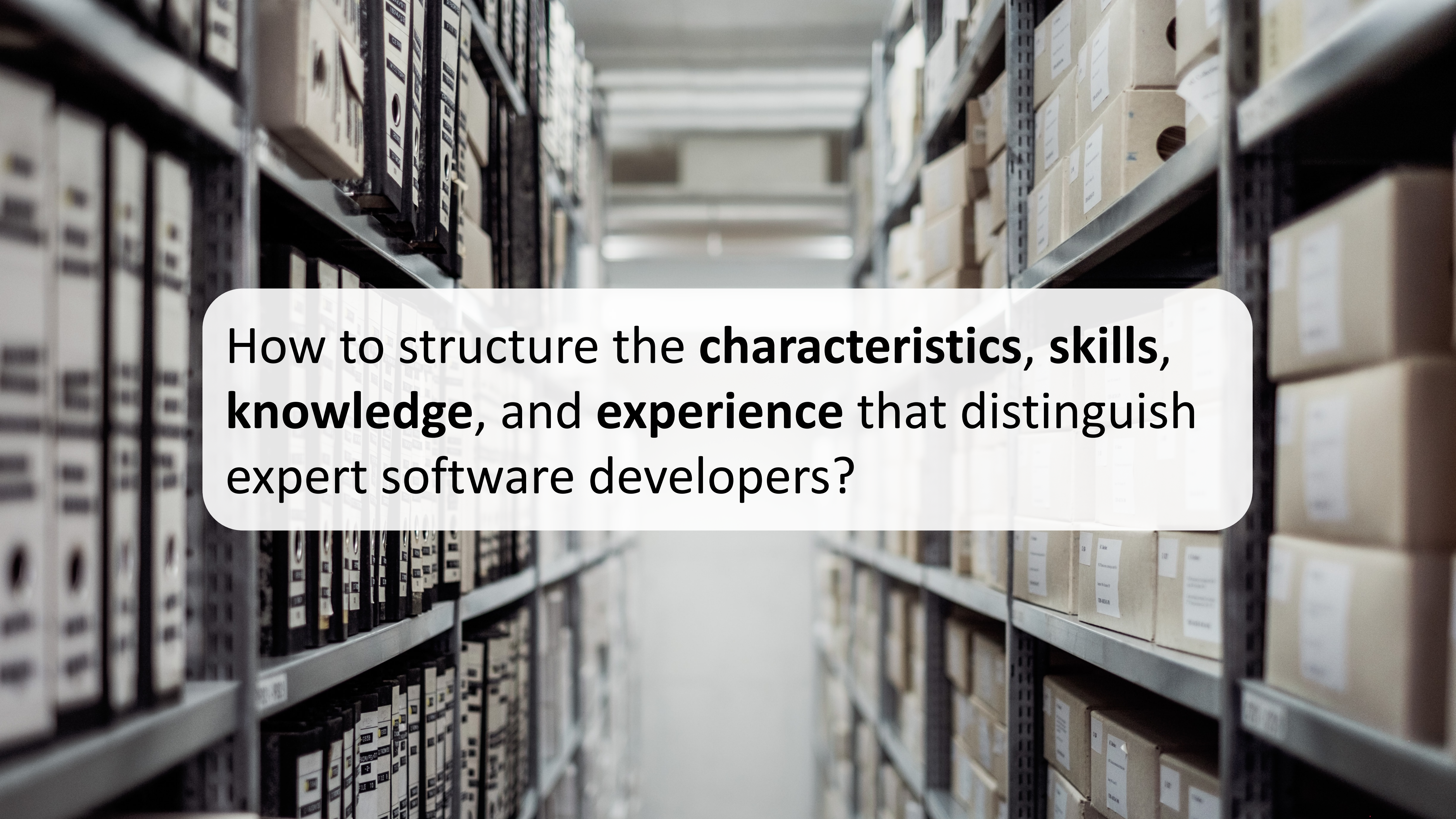
How are experience and expertise related?

Definitions

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

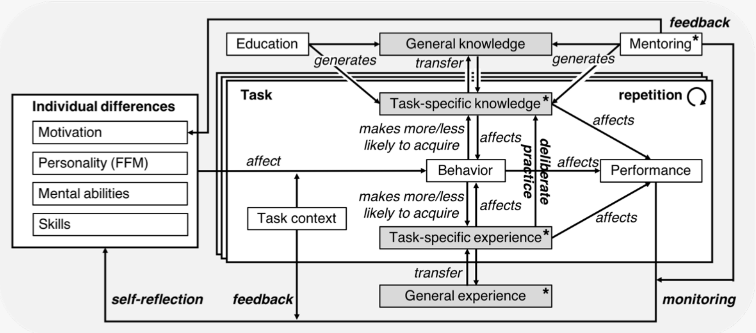
Expertise are *„the **characteristics**, **skills**, and **knowledge** that distinguish experts from novices and less **experienced** people."*

K. Anders Ericsson

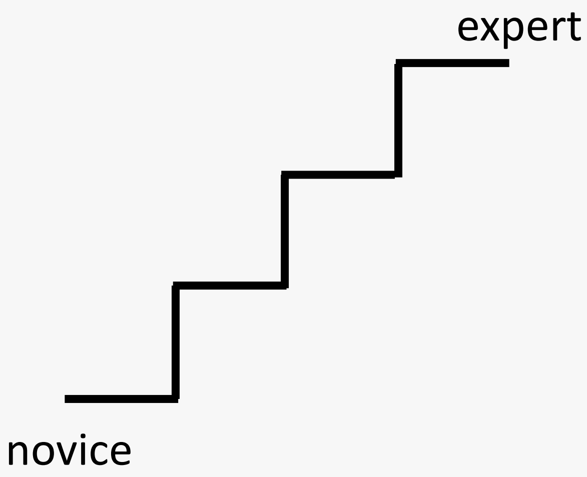


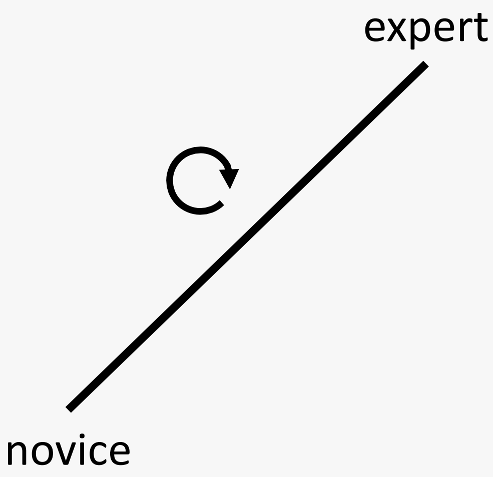
How to structure the **characteristics**, **skills**, **knowledge**, and **experience** that distinguish expert software developers?

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





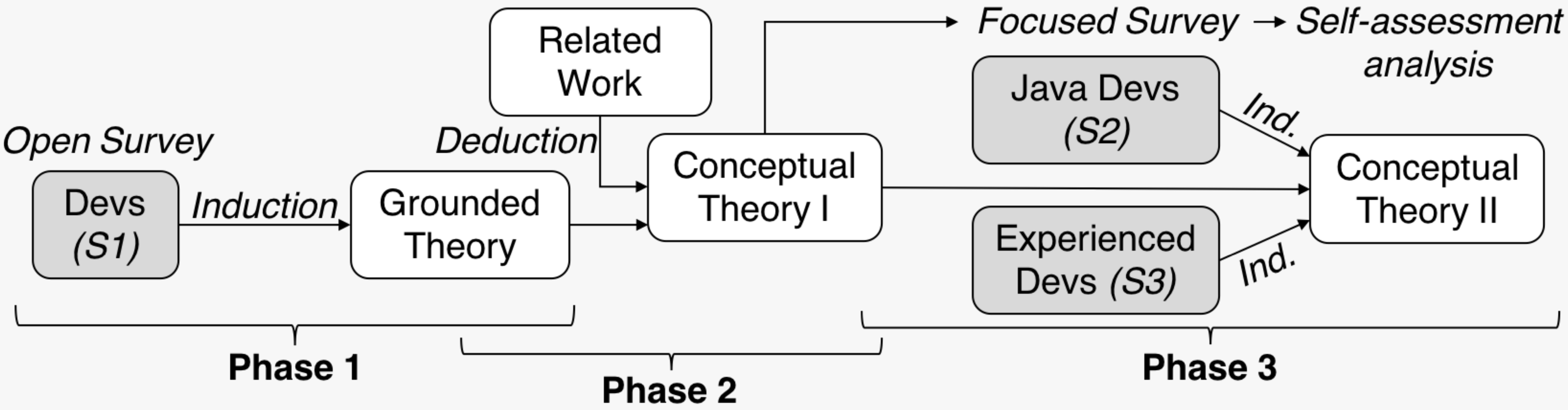
Theory Classification

- 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



Open Survey

Devs *(S1)*

Induction

Grounded Theory

Related Work

Deduction

Conceptual Theory I

Focused Survey → Self-assessment analysis

Java Devs *(S2)*

Experienced Devs *(S3)*

Ind.

Ind.

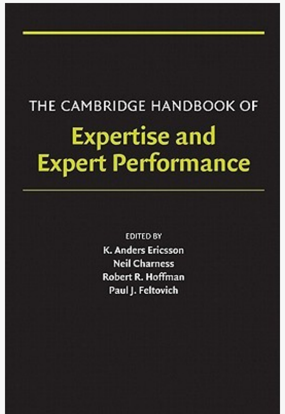
Conceptual Theory II

Phase 1

Phase 2

Phase 3

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



THE CAMBRIDGE HANDBOOK OF

Expertise and Expert Performance

EDITED BY

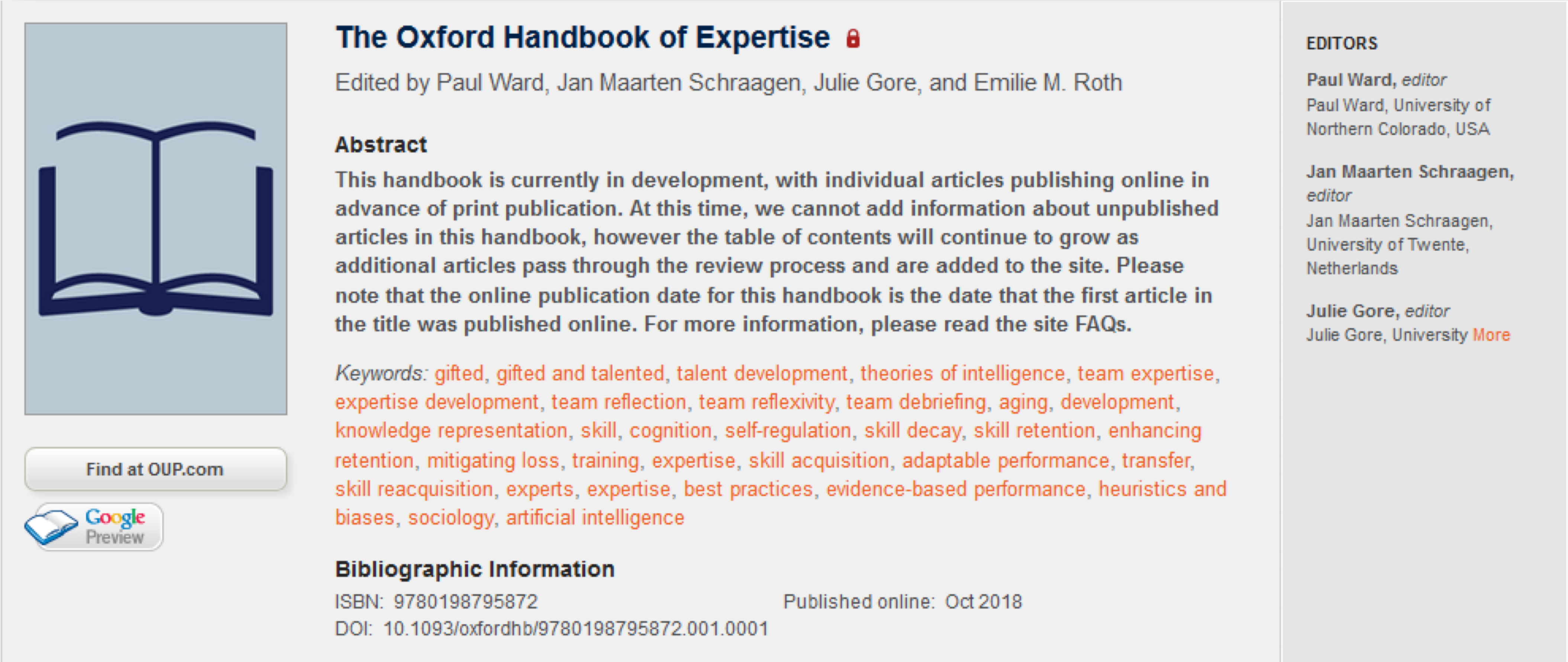
K. Anders Ericsson

Neil Charness

Robert R. Hoffman

Paul J. Feltovich

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	

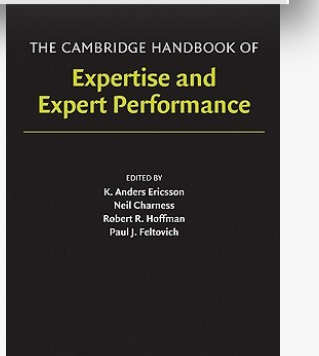
EDITORS

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

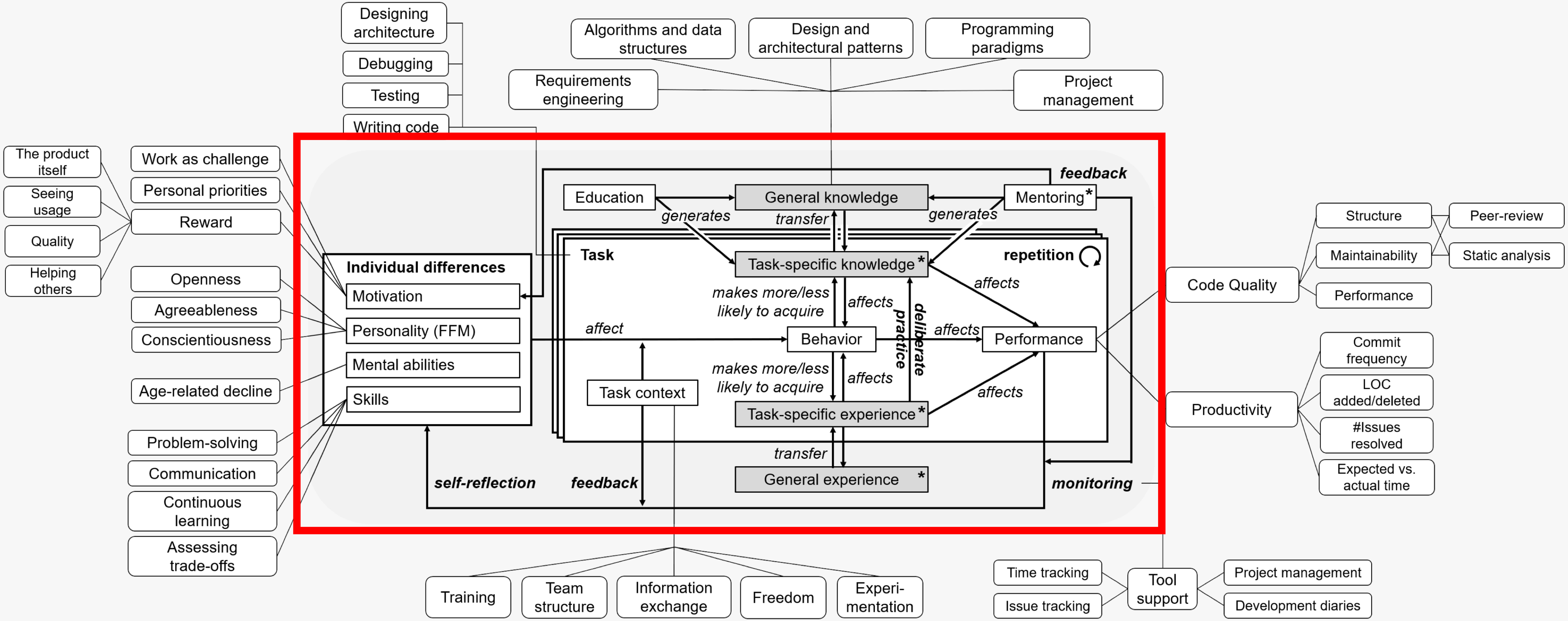
- **Induction:** 335 online survey participants in total
- **Deduction:** Main source *"Cambridge Handbook of Expertise and Expert Performance"*



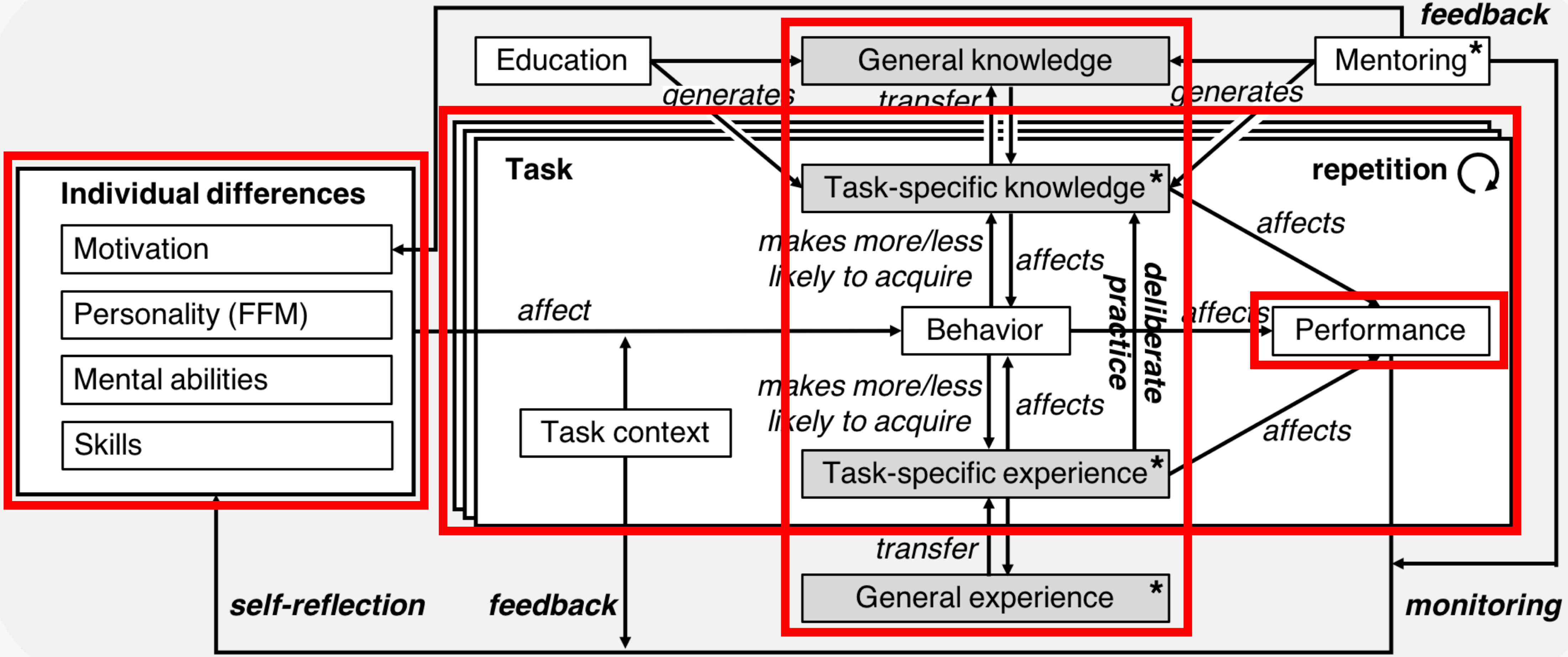
THE CAMBRIDGE HANDBOOK OF
Expertise and Expert Performance

EDITED BY
K. Anders Ericsson
Neil Charness
Robert R. Hoffman
Paul J. Feltovich

Final Conceptual Theory



Final Conceptual Theory

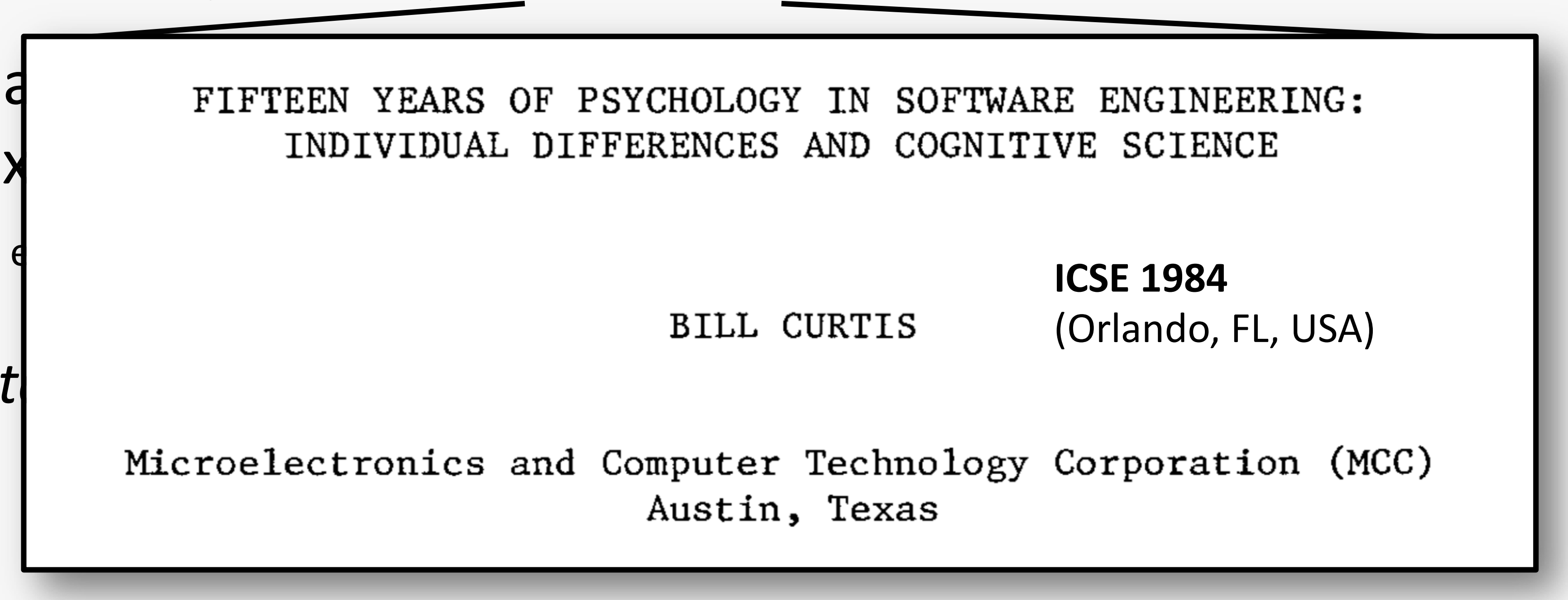


Knowledge

- **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 tha ***dependent****”,* but ex *and skills”* (Sonnentag e
- *“Semantic”* vs. *“synt*



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

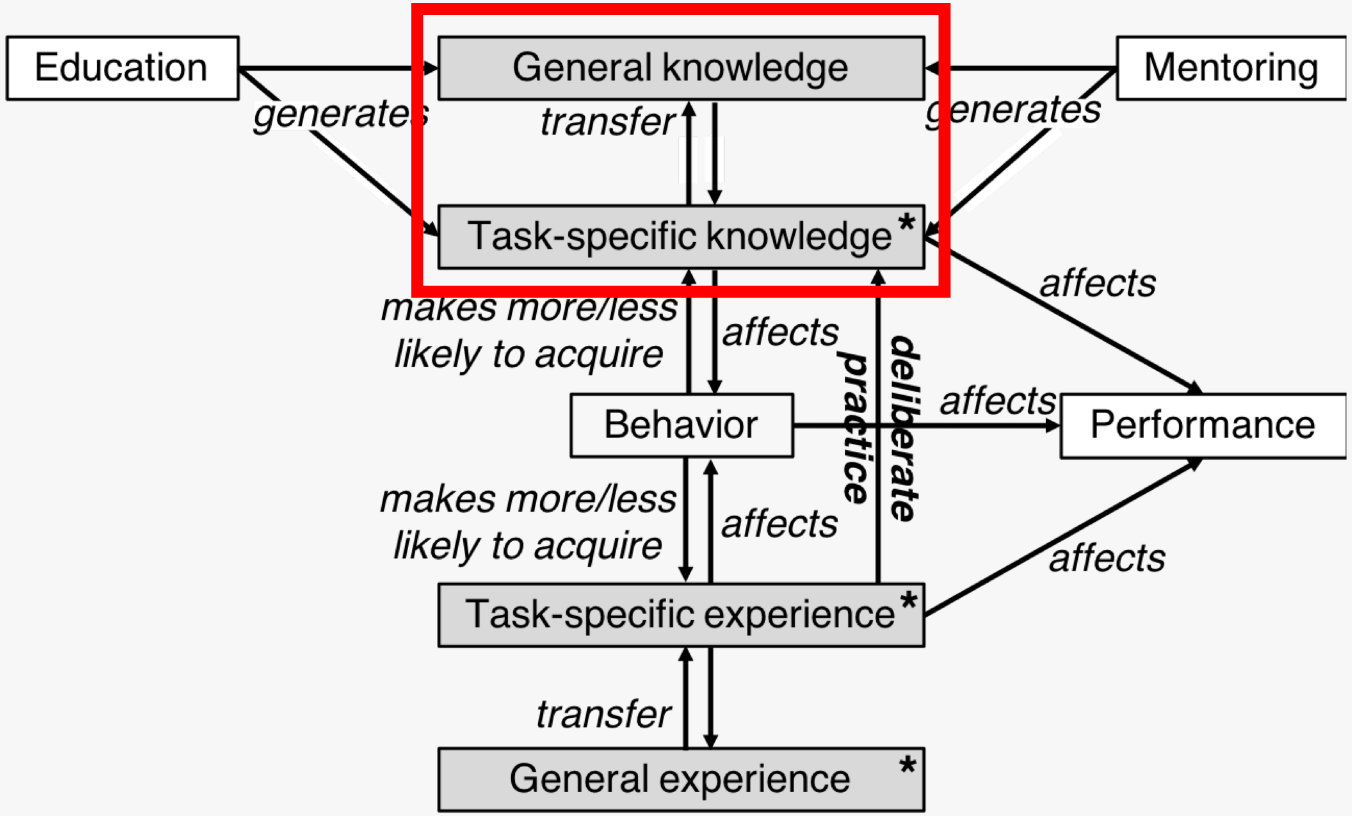
BILL CURTIS

Microelectronics and Computer Technology Corporation (MCC)
Austin, Texas

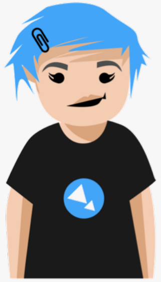
ICSE 1984
(Orlando, FL, USA)

Knowledge

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



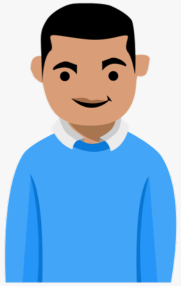
An *"intimate knowledge of the design and philosophy of the language"*



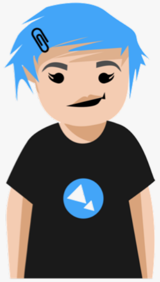
Experience

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

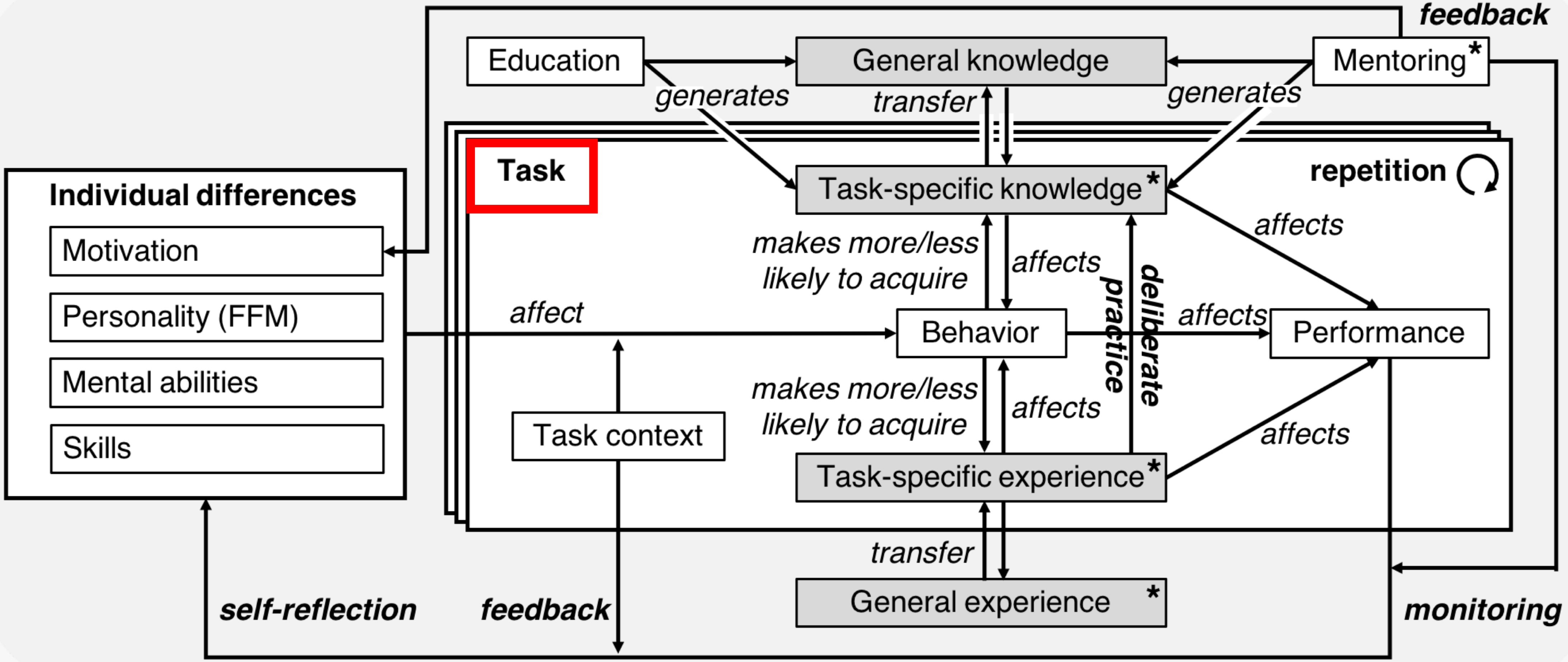
Having built *„everything from small projects to enterprise projects“*



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



Final Conceptual Theory



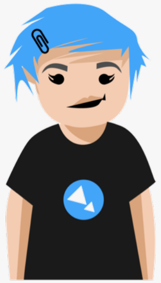
Tasks

- 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

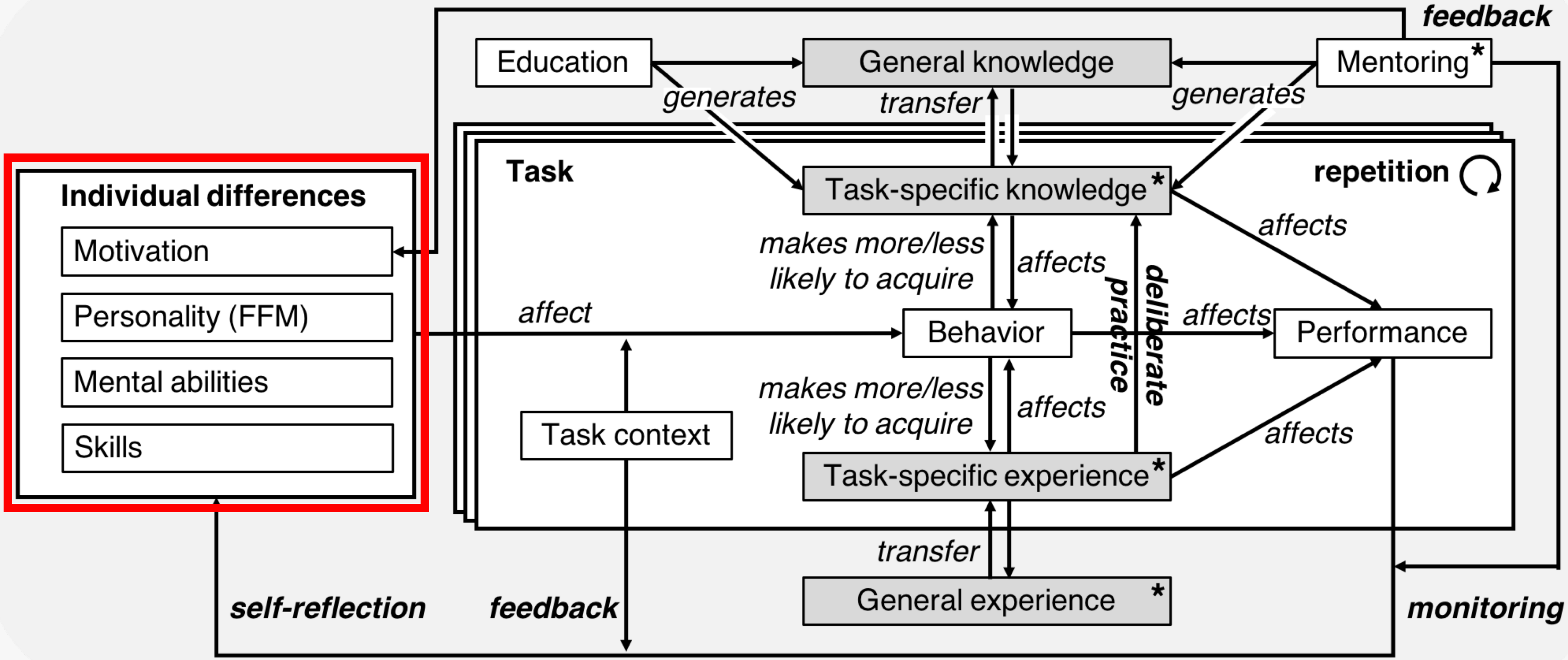
- Other mentioned tasks: testing, communicating, debugging

„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

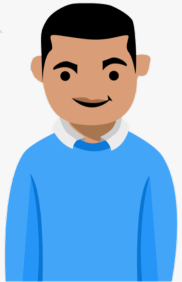


Individual Differences: Motivation

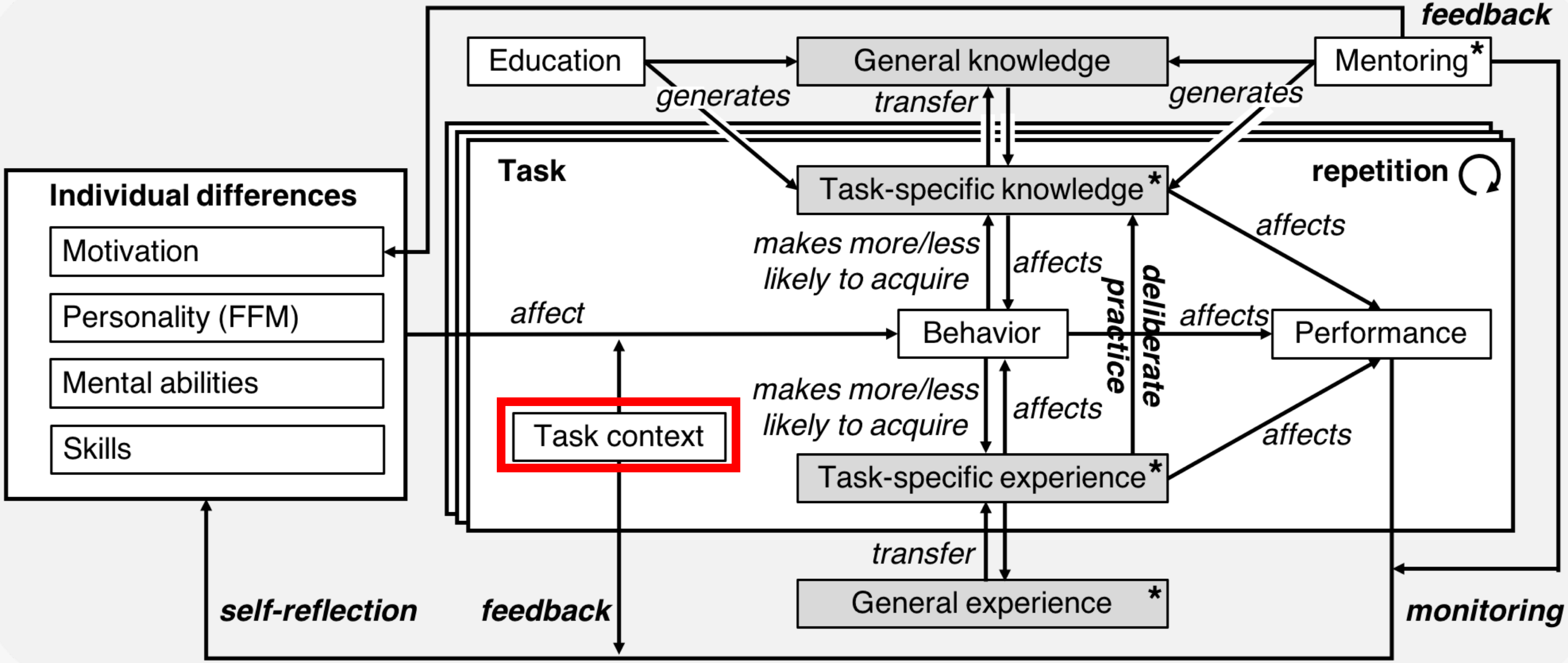
- Related work describes how **individual differences** affect expertise development
- Mental abilities and personality are relatively stable
- **Motivation can change** over time

- 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**."*



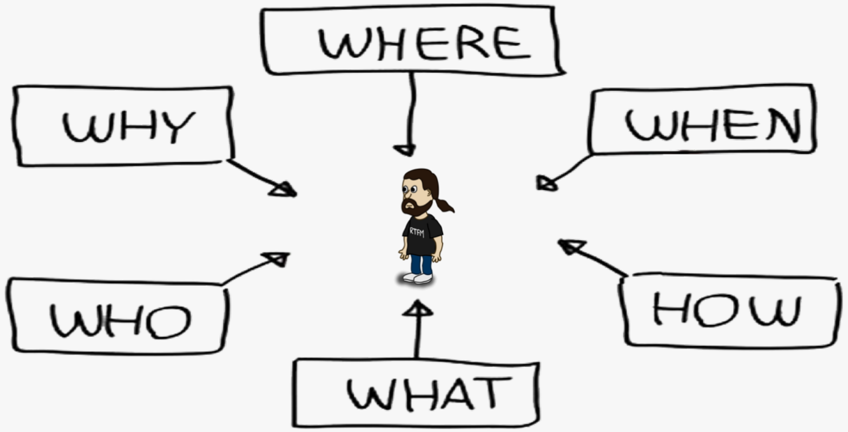
Final Conceptual Theory



Task Context

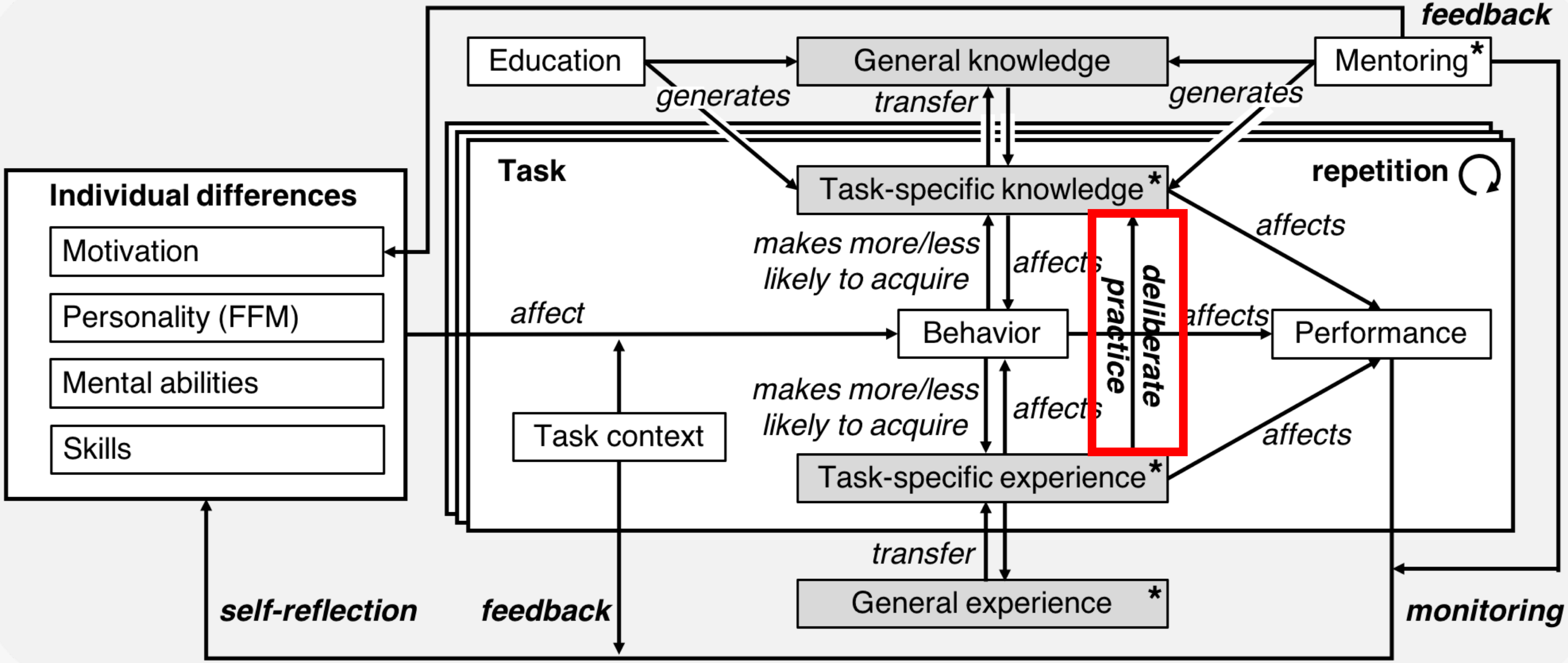
- Work **environment**
  (office, coworkers, customers etc.)
- Project **constraints**
  (external dependencies, time, etc.)
- Can either **foster or hinder** expertise dev.
- We asked: *What can employers do?*

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



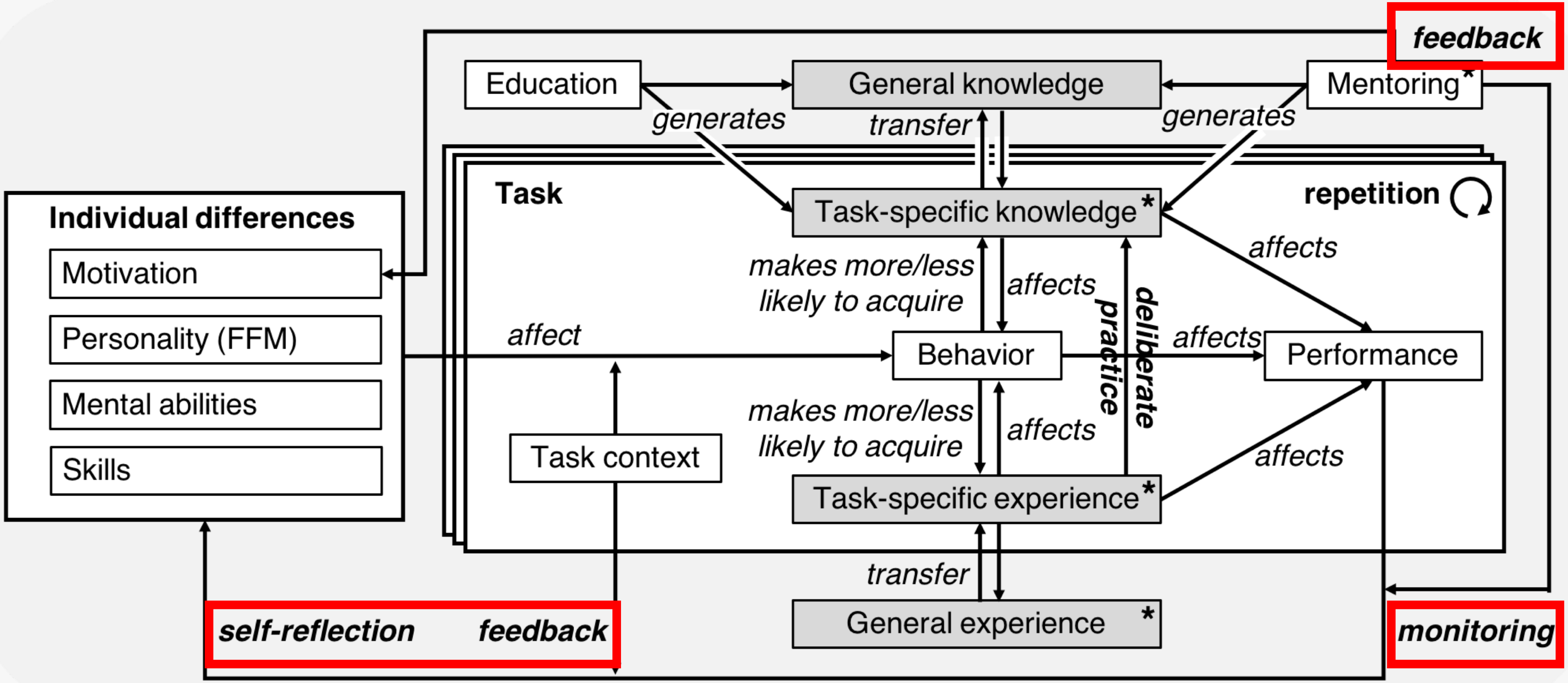
Deliberate Practice

- 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)

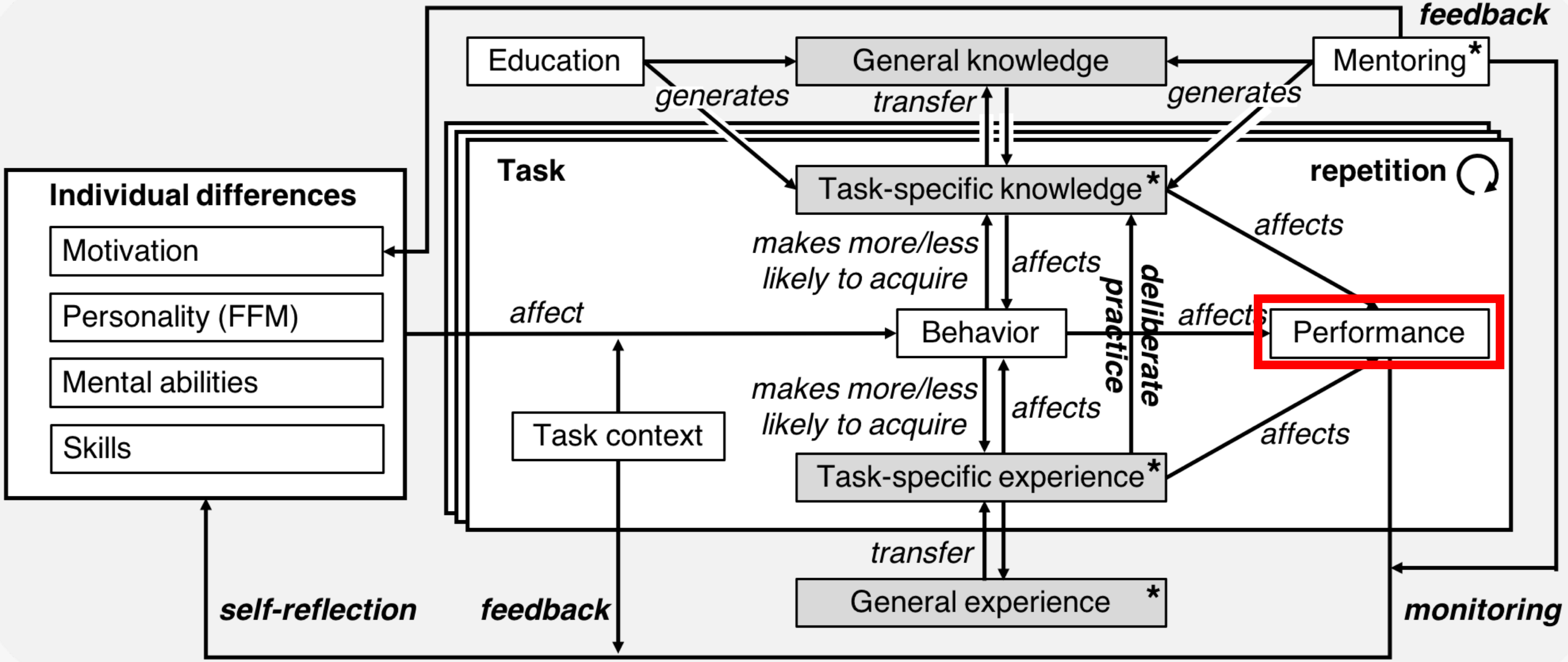
Deliberate Practice: Self-Reflection

- **(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

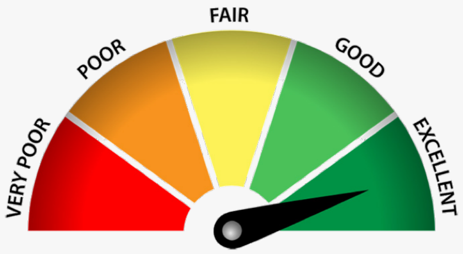
Final Conceptual Theory



Final Conceptual Theory



Performance



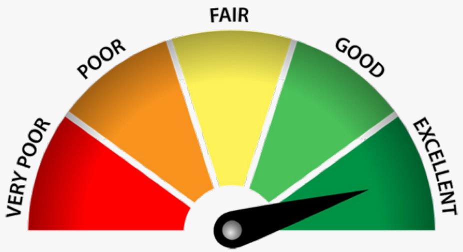
Scope of this work:

- 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:

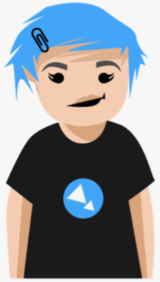
- Build **variance theory** for explaining and predicting the development of expertise

Performance

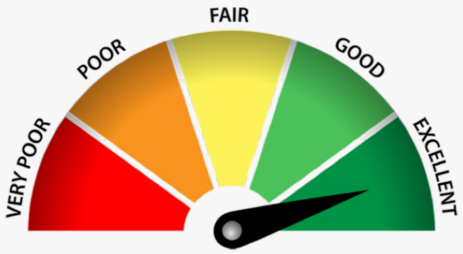


- Participants described different **properties of expert's source code** (well-structured, readable, maintainable, etc.)

„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

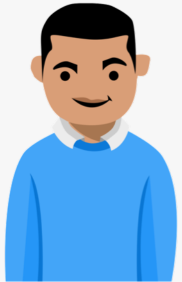
- Software development includes **many different tasks**

- 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

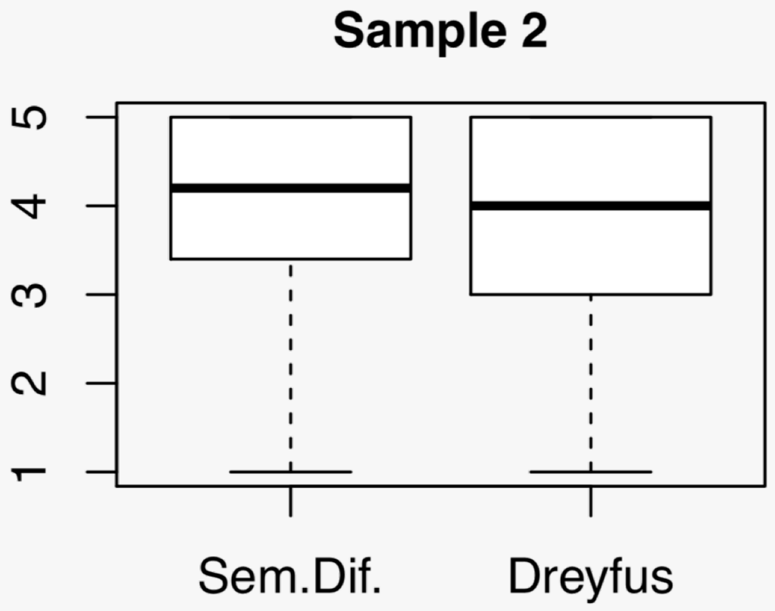
“Programming ability is based on **desire to achieve**. In the early years, it is a sort of **competition**. [...] I found that I lost a significant amount of my focus as I became 40, and started **using drugs such as ritalin** to enhance my abilities. This is pretty common among older programmers.’’

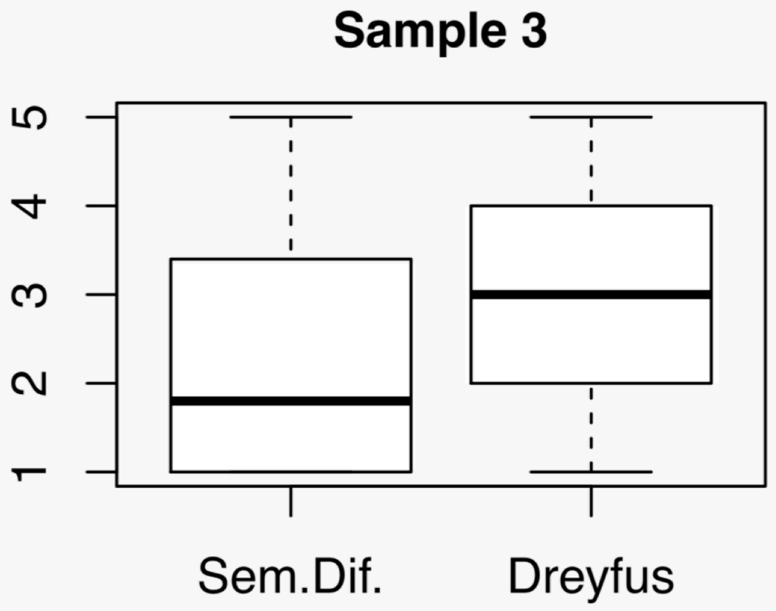
software developer, age 60

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

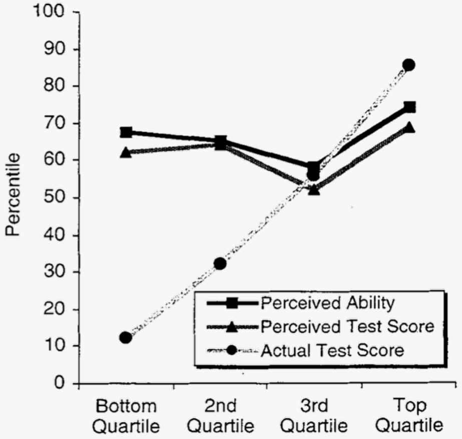




Experience vs. Expertise

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

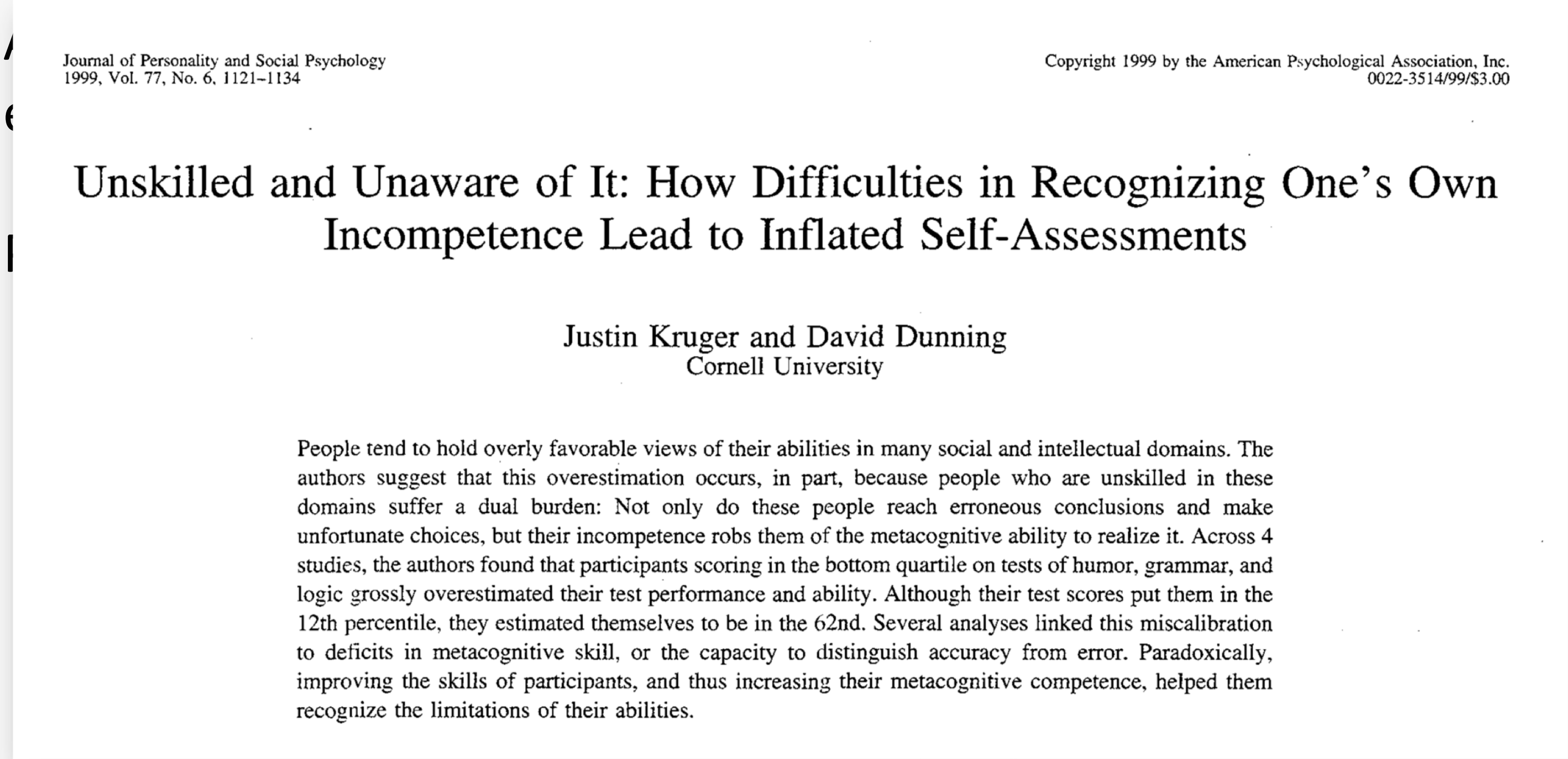
- Possible explanation: **Dunning-Kruger effect**
  - Participants with a high skill-level underestimate their ability and performance relative to their peers
  - Context helped experienced developers to adjust their ratings to be more accurate



Experience vs. Expertise

- A
  e

- I

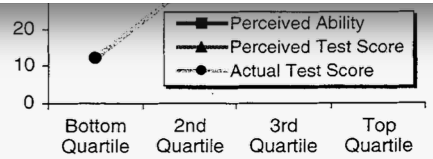


Journal of Personality and Social Psychology
1999, Vol. 77, No. 6, 1121–1134

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.

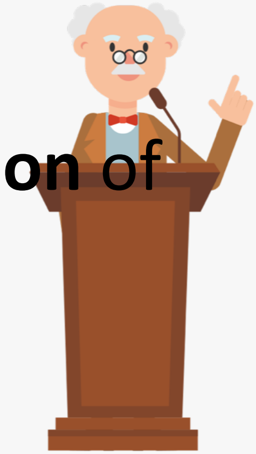


Takeaways

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



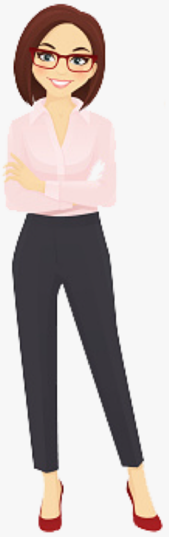
Summary for Developers

- 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

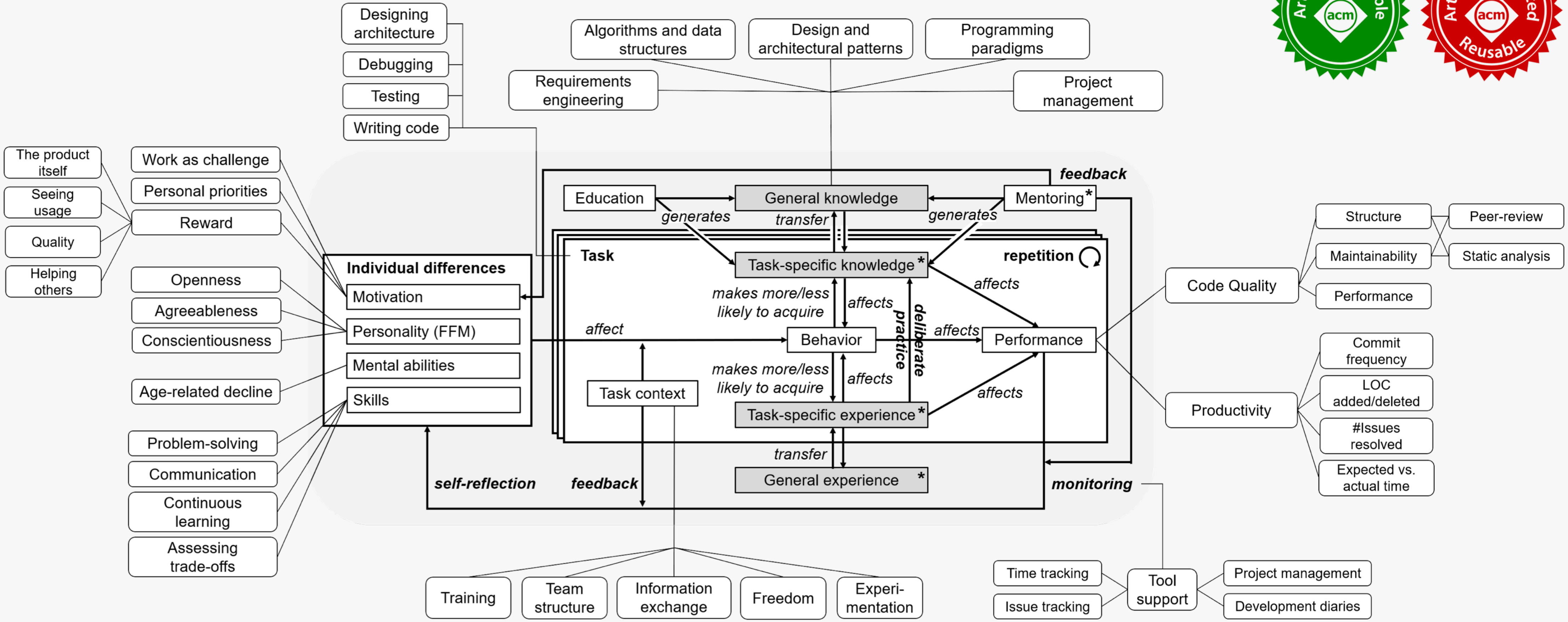
- Ideas on how to build supportive work environment **supporting self-improvement** of staff:
  - Good mix of continuity and change in software development process
  - Communicate clear visions, directions, and goals
  - Reward high-quality work wherever possible
  - Revisit information sharing in company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



Artifacts Available acm

Artifacts Evaluated acm Reusable

Artifacts Evaluated acm Functional

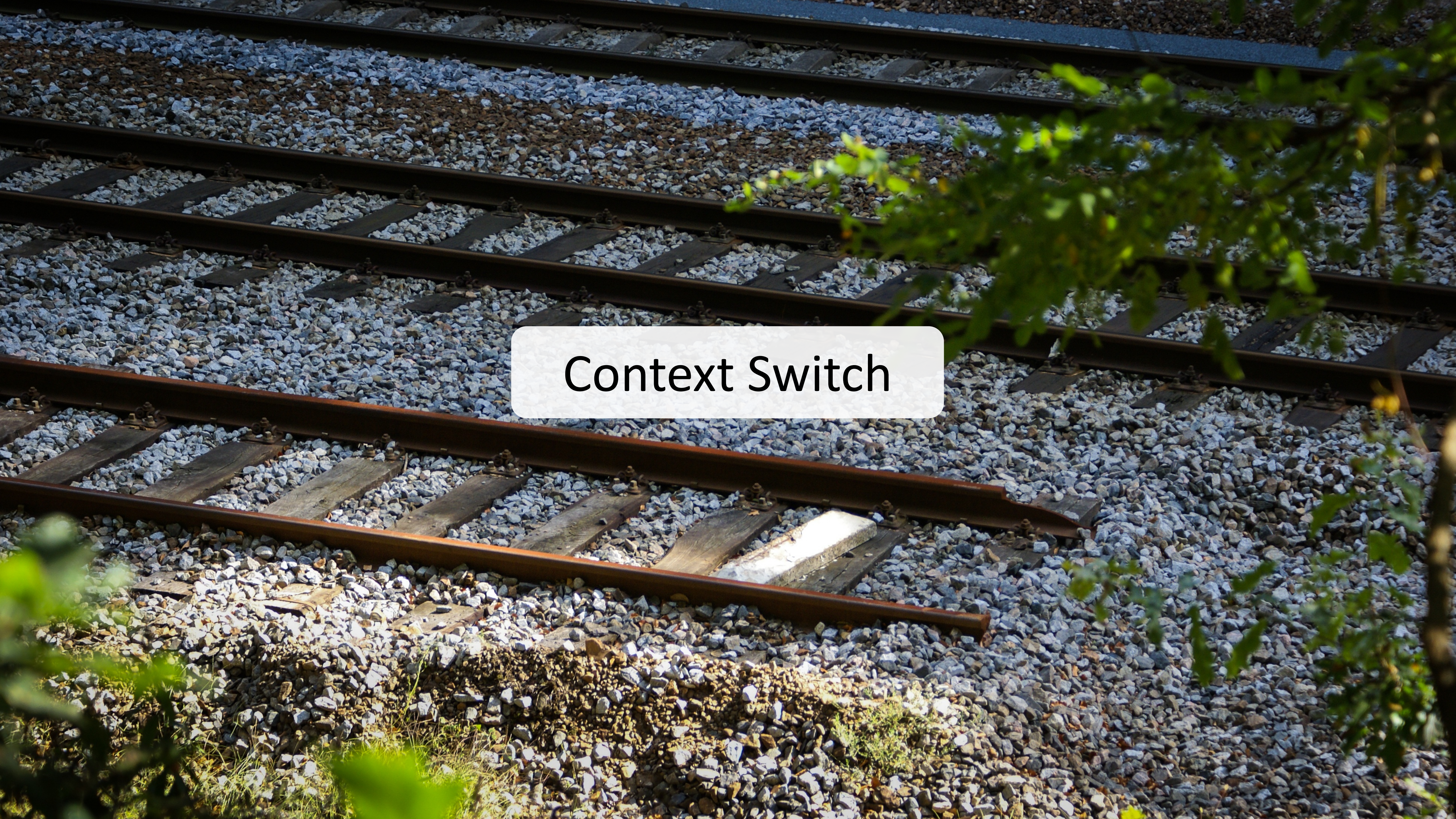


Sebastian Baltes

@s_balt
es

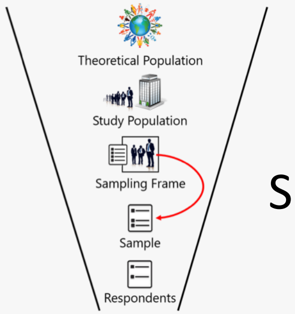
expertise.sbaltes.com

Data and scripts available on Zenodo



Context Switch

“Parallel Thread”



Issues in Sampling
Software Developers

Methodology

2013

Open Data

SOTorrent

airbnb

Constructing Urban
Tourism Space Digitally

Interdisciplinary Research

2018

SOTorrent

Studying the Origin, Evolution, and Usage of Stack Overflow Code Snippets

Sebastian Baltes

🐦 @s_baltes

sotorrent.org

Dataset available on Zenodo and BigQuery

Corresponding Research Papers

SOTorrent: Reconstructing and Analyzing the Evolution of Stack Overflow Posts

Sebastian Baltes
Lorik Dumani
research@sbaltes.com
dumani@uni-trier.de
University of Trier, Germany

Christoph Treude
christoph.treude@adelaide.edu.au
University of Adelaide, Australia

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

ABSTRACT

Stack Overflow (SO) is the most popular questio
site for software developers, providing a larg
snippets and free-form text on a wide variety o
software artifacts, questions and answers on S
for example when bugs in code snippets are fix
to work with a more recent library version, or
code snippet is edited for clarity. To be able to a
on SO evolves, we built *SOTorrent*, an open da
official SO data dump. *SOTorrent* provides acces
tory of SO content at the level of whole posts an
code blocks. It connects SO posts to other platfo
URLs from text blocks and by collecting refere

SOTorrent: Studying the Origin, Evolution, and Usage of Stack Overflow Code Snippets

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

Christoph Treude
University of Adelaide, Australia
christoph.treude@adelaide.edu.au

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

***Abstract*—Stack Overflow (SO) is the most popular question-and-answer website for software developers, providing a large amount of copyable code snippets. Like other software artifacts, code on SO evolves over time, for example when bugs are fixed or APIs are updated to the most recent version. To be able to analyze how code and the surrounding text on SO evolves, we built *SOTorrent*, an open dataset based on the official SO data dump. *SOTorrent* provides access to the version history of SO content at the level of whole posts and individual text and code blocks. It connects code snippets from SO posts to other platforms by aggregating URLs from surrounding text blocks and comments, and by collecting references from GitHub files to SO posts. Our vision is that researchers will use *SOTorrent* to investigate and understand the evolution and maintenance of code on SO and its relation to other platforms such as GitHub.**

dataset [16] that enables researchers to analyze the versio
history of SO posts at the level of individual text and co
blocks (see Figure 1 for exemplary posts). The official S
data dump [1] keeps track of different versions of e
posts, but does not contain information about diffe
between versions at a more fine-grained level. In partic
extracting different versions of the same code snippet from
the history of a post is challenging and required us to develop
a complex strategy, involving the evaluation of 134 different
string similarity metrics [15]. Besides providing access to the
version history, our dataset links SO posts to exte
in two ways: (1) by extracting linked URLs fro
of SO posts and from post comments and (2) by providing

MSR 2018/2019

MSR Mining Challenge 2019

Abstracts due Feb 1, 2019

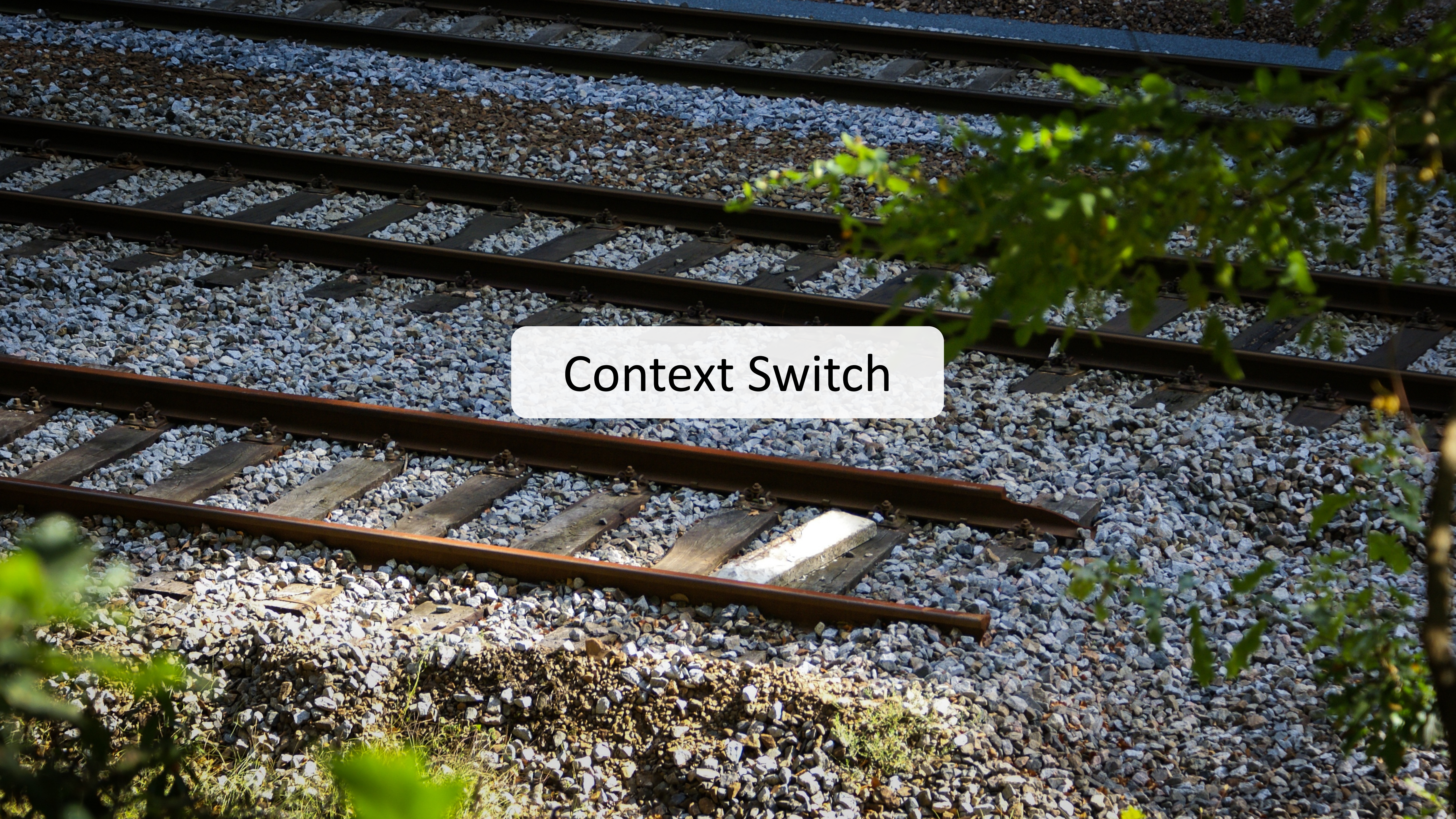
Papers due Feb 6, 2019

Sebastian Baltes

@s_balt
es

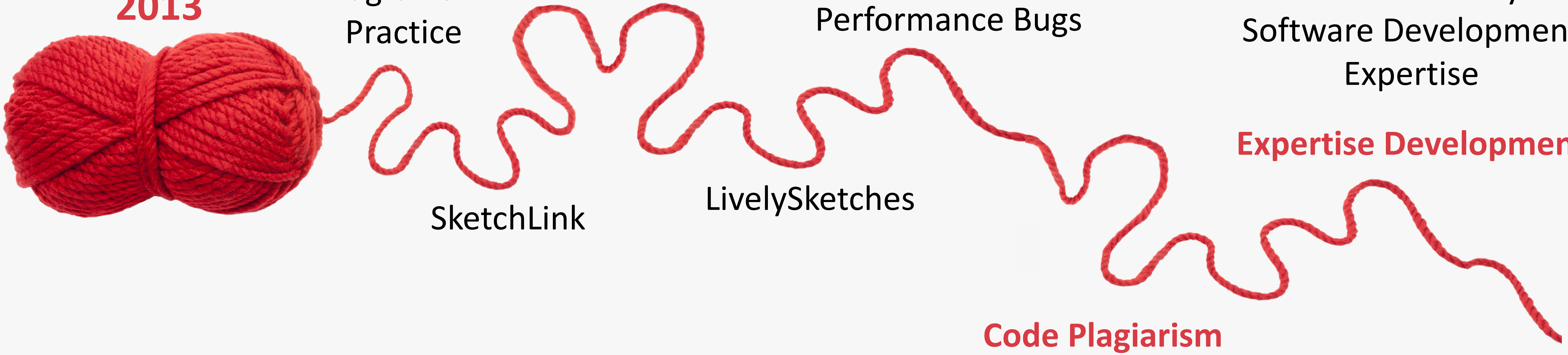
sotorrent.org

Dataset available on Zenodo and BigQuery



Context Switch

Studied Habits



2013

Sketches and Diagrams in Practice

Sketching

SketchLink

LivelySketches

How Developers Locate Performance Bugs

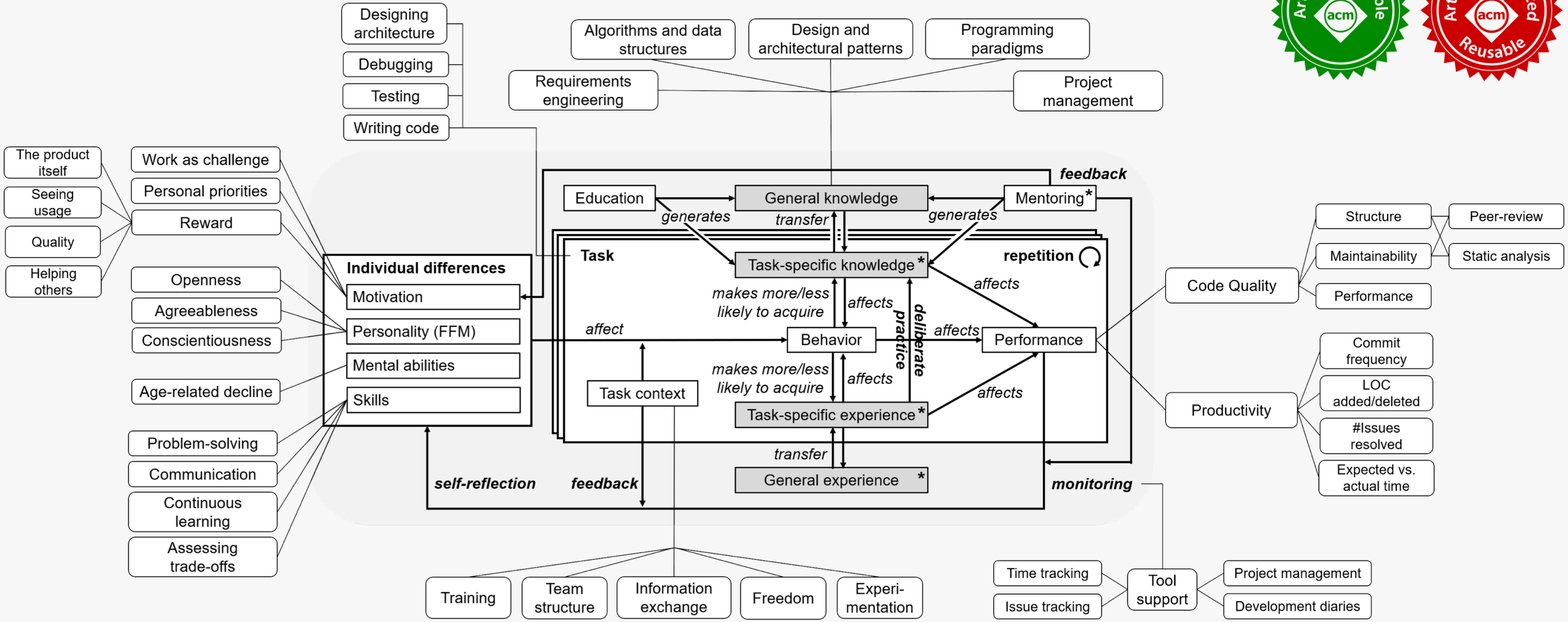
Code Plagiarism

stackoverflow Code Snippets in GitHub Projects

Towards a Theory of Software Development Expertise

Expertise Development

2018



Sebastian Baltes

@s_balt
es

expertise.sbaltes.com

Data and scripts available on Zenodo