Evaluation Bias and Epistemic Inequality in Global Software Development

Sam Khosravi
KTH Royal Institute of Technology
Stockholm, Sweden
samkh@kth.se

Amir H. Payberah
KTH Royal Institute of Technology
Stockholm, Sweden
payberah@kth.se

Abstract

This paper examines fairness and accountability in global software development by focusing on how competence is assessed and valued across unequal regional contexts. We compare software engineers from East Africa (Rwanda and Uganda) and Northwestern Europe (Sweden and the Netherlands), regions that are increasingly connected but embedded in asymmetric technological, economic, and institutional structures. Despite the rapid growth of African technology ecosystems, empirical evidence on everyday engineering practice and evaluation in these contexts remains limited. We present findings from an on-site mixed-methods study with 48 software engineers across four countries. The study combines programming, system design, and code review tasks with semi-structured interviews. Our results reveal consistent gaps between measured performance and perceived competence. Senior engineers in Rwanda often performed at a level comparable to that of their European peers, yet European participants systematically underestimated the competence of East African engineers. We also observed differences in communication styles and organizational practices across regions, reflecting distinct but complementary ways of working.

CCS Concepts

• **Human-centered computing** → *Collaborative and social computing*; *Ethnographic studies.*

Keywords

Global Software Development, Evaluation Bias, Epistemic Unfairness, Cross-Cultural Collaboration, Technical Competence

1 Introduction

Global software development increasingly relies on collaboration across regions with unequal access to resources, infrastructure, and professional recognition. In these settings, assessments of competence shape who is trusted, whose contributions carry weight, and who gains access to career opportunities. Engineers in historically marginalized regions are often evaluated through stereotypes and deficit narratives that frame their work as lower quality or higher risk. These evaluations are not merely individual misunderstandings; they constitute a fairness and accountability problem in the global computing labor market. By shaping whose knowledge is treated as credible and whose expertise is discounted, they contribute to epistemic injustice and procedural unfairness in everyday sociotechnical work practices.

East Africa illustrates this tension clearly. Countries such as Rwanda and Uganda have invested heavily in digital skills and software engineering capacity and are increasingly positioned as sites of global technology work, while Northwestern European ecosystems such as Sweden and the Netherlands remain comparatively well-resourced and institutionally entrenched. Studying these regions together highlights how competence is assessed and compared across unequal contexts. In cross-regional collaboration, perceptions of skill can serve as gatekeeping mechanisms that shape whose work is trusted and whose careers advance, allowing inequitable distributions of credibility and opportunity to persist even when technical evidence suggests overlapping capability.

Despite the relevance of these dynamics, prior work on global software engineering has focused largely on Asia, Eastern Europe, and South America, leaving African contexts underexplored [36, 40, 48, 52]. Existing studies on Africa concentrate mainly on South Africa and Kenya [26, 65, 73, 74], while countries such as Rwanda and Uganda remain empirically underrepresented despite major digital infrastructure initiatives [72]. Moreover, much of this literature relies on secondary data, limiting insight into how perceptions and organizational narratives relate to observed engineering practice [4]. As a result, an important dimension of fairness in computing remains insufficiently examined: how human and organizational evaluation systems distribute trust, credibility, and opportunity across unequal global labor contexts.

To address this gap, we study software engineers in Rwanda, Uganda, Sweden, and the Netherlands to examine how technical performance, communication practices, and competence perceptions interact across unequal contexts. Our analysis is guided by the following research questions:

- RQ1: How do software engineers in East Africa and Northwestern Europe differ in coding skills and problem-solving strategies?
- RQ2: How do communication styles and organizational practices differ between regions, and how are these differences interpreted in cross-regional collaboration?
- RQ3: What forms of perception bias and procedural unfairness arise in cross-regional software development, and how do these biases affect the recognition of competence and the distribution of trust?

We adopt a pragmatic mixed-methods approach combining controlled technical assessments with qualitative inquiry. We conducted on-site sessions with 48 software engineers (10-15 per country), who completed a programming task, a system design challenge, and a code review exercise. Programming solutions were analyzed using CodeBERT embeddings [21], PCA [1], and K-means clustering [68], complemented by manual inspection, while review-comment sentiment was assessed using VADER [35]. Semi-structured interviews (60-120 minutes) captured developers' self-assessments, perceptions of peers, and experiences of cross-regional collaboration.

Our findings reveal systematic gaps between measured performance and perceived competence. Senior Rwandan engineers often performed on par with European peers, while junior performance varied more widely. Communication styles differed consistently: East African engineers tended to provide more supportive feedback, whereas European engineers favored more direct critique. Organizational emphases also diverged, with European participants prioritizing upfront design and East African participants emphasizing review and deployment. Crucially, European engineers consistently underrated East African capabilities despite evidence of overlapping competencies, indicating that evaluation practices in global software development can reproduce epistemic and procedural unfairness even when technical contributions warrant recognition.

Contributions. This paper contributes:

- Empirical evidence from underrepresented contexts (Rwanda and Uganda) on how competence, credibility, and professional recognition are constructed in global software development.
- A comparative mixed-methods analysis linking technical assessments, communication practices, and organizational priorities across East Africa and Northwestern Europe.
- Documentation of perception bias and procedural unfairness, showing systematic underrating of East African engineers by European peers despite demonstrated overlap in capability.
- Implications for more accountable global collaboration, including how evaluation and feedback practices might be redesigned to reduce bias and better recognize complementary strengths.

The remainder of the paper reviews related work, presents the study design and methods, reports results, and discusses implications and limitations.

2 Background and Related Work

This section provides the theoretical and methodological context for our study. It begins by discussing the evolution of global software engineering and the role of cross-cultural collaboration, followed by an overview of analytical techniques and methodologies relevant to evaluating software engineering skills and practices.

2.1 Global Software Engineering and Cross-Cultural Collaboration

Global software engineering has evolved from co-located, in-house teams to geographically distributed collaborations that span continents. Advances in Information and Communication Technology (ICT), cloud-based tools, and agile practices have enabled organizations to recruit talent worldwide and coordinate work across time zones [16, 18, 19, 83]. Economic drivers, such as rising labor costs in established markets and the pursuit of cost efficiency and specialized expertise, have further incentivized outsourcing and offshoring [20].

While distributed collaboration offers these benefits, it also introduces challenges of distance, typically categorized as *geographical*, *temporal*, and *socio-cultural* [66]. Geographical distance reduces informal communication and shared awareness; temporal distance limits synchronous interaction and delays feedback; and socio-cultural distance introduces differences in work practices, hierarchies, and communication styles. Together, these factors can weaken trust, complicate coordination, and reduce productivity [34, 70]. Although agile methodologies and digital tools mitigate some barriers, temporal and cultural differences remain persistent challenges [39, 82].

Cultural variation plays a central role in shaping collaboration in global software development. Hofstede's cultural dimensions [33, 46] highlight contrasts in power distance, uncertainty avoidance, and individualism vs. collectivism. For example, high-context communication common in collectivist societies can clash with low-context communication in individualist cultures, often leading to misunderstandings [47, 61, 70]. Studies also show that African and European teams differ in communication and organizational practices, with African teams more often operating within hierarchical structures and European teams favoring flatter ones [76].

These differences are often compounded by biases, which shape how competence and contributions are perceived and can reinforce inequalities in distributed teams [45, 50, 81]. However, interventions such as cultural competence training and support for multilingual collaboration can help mitigate these barriers [59, 80]. Despite this need, most empirical studies on cross-cultural software engineering and outsourcing concentrate on Asia, Eastern Europe, and South America [5, 36, 40, 64]. A systematic review of 91 outsourcing papers across 51 journals found no Africa-based studies [30], and the limited research on Africa largely examines South Africa, Kenya, Nigeria, or North Africa [26, 65, 73, 74].

Against this backdrop, Rwanda and Uganda are emerging as notable technology hubs in East Africa, driven by large-scale ICT investments, broadband expansion, and supportive policies [25, 54, 55, 69, 75]. Rwanda, in particular, has positioned ICT as a national growth engine, with the sector contributing to a 35% increase in Gross Domestic Product (GDP) in 2023 [75]. Entrepreneurial ecosystems, such as Kigali's Norrsken hub, further signal international attention to the region's potential [8, 56, 57, 60]. Yet empirical research on software engineering practices in these countries remains scarce, leaving Rwanda and Uganda largely absent from comparative global software engineering studies. This underrepresentation motivates our study to generate new insights into how cultural and organizational contexts shape software practices in underexplored regions.

2.2 Analytical Techniques and Methodologies in Software Engineering Research

Software engineering research employs a wide range of methods to evaluate both technical competencies and collaborative practices. Quantitative assessments such as standardized coding tests, bug detection tasks, and system design exercises provide objective measures of programming proficiency, algorithmic thinking, and problem-solving strategies [17, 23, 24, 63]. Qualitative approaches, including in-depth interviews and field studies, complement these metrics by capturing organizational norms, cultural dynamics, and perceptions of collaboration [43, 51]. Mixed-methods designs are often recommended for their ability to balance objective measurement with contextual richness [9, 62].

Building on these foundations, recent advances in machine learning and natural language processing have significantly expanded the analytical toolkit for studying software practices at scale. Transformer based models such as CodeBERT [21], GraphCodeBERT [28], CoCluBERT [29], PLBART [2], and CodeT5/CodeT5+ [78, 79] generate code embeddings that enable embedding, retrieval, and clustering of programming solutions and styles. Dimensionality reduction techniques, such as PCA [1], further support the discovery and visualization of coding patterns in high-dimensional spaces [49]. For collaborative practices, sentiment and affective computing approaches have been applied to pull requests, issue discussions, and code reviews; for example, classifiers such as Senti4SD [10] and SentiCR [3], along with lexicon-based tools like VADER [35, 58], have been used to examine tone, supportiveness, and directness in developer communication. Together, these methods enable a holistic assessment of software engineering, capturing technical performance, design reasoning, communication styles, and cultural perceptions.

3 Method

This study adopts a pragmatic mixed-methods approach [71], combining quantitative coding assessments with qualitative interviews to compare software engineers in East Africa (Rwanda and Uganda) and Northwestern Europe (Sweden and the Netherlands). Pragmatism enables the balancing of measurable outcomes with contextual insights, avoiding the rigidity of purely quantitative or qualitative paradigms [12, 22, 41].

3.1 Research Questions

The study is guided by three research questions (RQ), which are formulated based on gaps identified in the literature regarding global software engineering and cross-cultural collaboration with particular attention to the limited representation of East Africa contexts [37].

RQ1: How do software engineers in East Africa and Northwestern Europe differ in coding skills and problem-solving strategies? We approached this question with quantitative methods, including standardized coding and system design tests that measure programming skills and solution complexity. We analyzed the results using machine learning and clustering techniques for comparative insights. In addition, we conducted in-depth on-site interviews to explore the reasoning behind the strategies.

RQ2: How do communication styles and organizational practices differ between regions? We explored this question by applying sentiment analysis to code comments and expressions used when solving problems. Quantitative data highlight differences in process steps across countries, while qualitative interviews with team leads provide insights into organizational cultures.

RQ3: What perception biases exist in software development across these regions, and how might they influence collaboration between engineers? We investigated this question through a perception study that quantitatively assesses biases and through conversations with engineers about their responses. At later stages, we asked interviewees to reflect on the results of the perception study.

3.2 Data Collection

This study draws on data from 48 software engineers, with 10–15 participants each from Rwanda, Uganda, Sweden, and the Netherlands. Participants represented different levels of seniority, with *juniors* defined as having under three years of experience and *seniors* as having over three years, to capture variation in experience and perspective. Each session was conducted on-site in Rwanda, Uganda, Sweden, and the Netherlands, lasting 60–120 minutes and covering coding, system design, and code review tasks, followed by an in-depth interview. To ensure comparability, participants could use a programming environment of their choice, but they were not permitted to access the internet or AI tools.

The sample size was carefully chosen to ensure validity. Prior research shows that data saturation in qualitative studies is typically reached after 9–17 interviews within homogeneous groups [15, 31], and that most themes are identified within the first dozen interviews [27]. With 10–15 participants per country, this study ensures saturation at the national level, while 48 interviews in total provide breadth for cross-cultural comparison. Recruitment relied on professional networks through convenience sampling [67], a pragmatic choice in exploratory cross-cultural research where access can be limited. This approach supports the study's goal of capturing authentic perspectives from practicing engineers across regions. Software engineers share core training and practices, providing a relatively homogeneous group, while the inclusion of four countries adds the heterogeneity needed for comparative analysis.

To identify comparative patterns of skills, practices, and perceptions across regions, we employed a mixed-methods design. Quantitative analyses highlight broad trends, while qualitative interviews provide contextual depth for interpreting those patterns [14, 38]. Prior work emphasizes that even when quantitative precision is limited, such indicators remain valuable for triangulation when complemented with rich qualitative insights [32]. To strengthen validity, all tasks and interview questions were grounded in real-world engineering practices. They were designed using competencies from the scientific literature and refined with feedback from experienced practitioners.

The tasks also covered six of the ten SWEBOK knowledge areas, including software design, construction, testing, quality, process, and maintenance, ensuring coverage of core engineering skills [13]. Reliability was supported by administering identical tasks under the same conditions to all participants, applying predefined scoring criteria, and conducting pilot testing to confirm clarity. To safeguard integrity, coding and design tasks were monitored to ensure independent work, and in virtual interviews, a second observer was present to validate the process.

3.3 Assessments

We designed three tasks to evaluate both technical and communication skills, complemented by a study of development process estimates. Together, these assessments targeted three dimensions: technical skills, communication styles, and perceptions.

Task1: Programming Assignment (Unique Pairs to Target). To evaluate algorithmic thinking and coding efficiency, participants solved a variation of the well-known two-sum problem [42]. Given a list of integers and a target value, they were asked to return all unique pairs of numbers that summed to the target. For example:

```
Input: nums = [1, 5, 3, 7, 4], target = 8
Output: [(1, 7), (5, 3)]
Input: nums = [2, 2, 4, 4], target = 8
Output: [(4, 4)]
```

The task is simple enough to complete within a short time yet complex enough to differentiate skill levels. Participants typically used one of three strategies: (1) the *brute-force* approach that checked all possible pairs in $O(n^2)$ time, which is easy to implement but inefficient; (2) the *sorting and two-pointer technique* that first sorted the list and then used two indices moving from opposite ends to find pairs in $O(n \log n)$ time; and (3) the most efficient solution, the *hash set method*, that tracked seen numbers and identified complements in a single pass in $O(n)$ time.

To analyze submissions, we used CodeBERT [21] to generate embeddings of the code, reduced them with PCA [1], and applied K-means clustering [68] to identify structural patterns. This automated analysis was complemented by manual inspection, which classified whether participants used brute force, two-pointer, or hash set solutions, and noted their reasoning during interviews. Together, these analyses provided both quantitative and qualitative insights into regional problem-solving strategies.

Task 2: System Design Challenge (URL Shortener). To evaluate architectural design, scalability, and data handling, participants completed a system design challenge to outline a URL shortening service similar to `Bit.ly`. The task required them to ensure efficient request handling, unique short links, reliable storage and retrieval, and accurate redirection. Participants explained their solutions verbally in real time, which enabled the assessment of both technical reasoning and communication clarity.

This challenge reflects practical, real-world scenarios and provides a fair basis for comparing engineers across regions. Prior studies show that software engineers often lack practical design skills, particularly in scalability and architectural reasoning [77]. This task addresses this skills gap by examining whether participants could design a scalable and maintainable system. The results were aggregated manually by reviewing each answer and identifying overall country-level patterns, which are presented in Section 4.

Task 3: Code Review Challenge (Authentication System). To evaluate both security awareness and feedback practices, participants reviewed a deliberately flawed authentication system and proposed improvements via code comments. This task assessed vulnerability detection and the clarity/constructiveness of review feedback, enabling cross-regional comparison. Code reviews improve software quality [7], and including security defects reflects expectations that developers contribute to secure software even when not security specialists [6]. Listing 1 shows the code provided to participants for review.

```
# This authentication system provides basic login and
# password management functionality

class AuthSystem:
    def __init__(self):
        # Hardcoded credentials and plaintext password
        # storage (insecure)
        self.users = {"admin": "password123"}
        # TODO: replace with secure storage

    def login(self, username, password):
        # Unsafe equality instead of hashed comparison
        if (username in self.users and
                self.users[username] == password):
            print("Login successful!")
            return True

        # No rate limiting or account lockout
        return False

    def change_password(self, username, new_password):
        # No authentication or authorization check
        # No strength validation, hashing, or audit log
        self.users[username] = new_password

# Example usage
auth = AuthSystem()
auth.login("admin", "password123")
auth.change_password("admin", "newpass")
auth.login("admin", "newpass")
```

Listing 1: Insecure authentication system provided to participants for the code review challenge.

Participants identified common flaws, including plaintext password storage, hardcoded credentials, no authentication for password changes, weak error handling, missing validation, and a lack of rate limiting. We categorized 11 vulnerability issues in total and compared how frequently juniors and seniors in each region identified them. The participant comments referring to vulnerability categories are classified as *positive* (supportive and constructive), *neutral* (descriptive and technical), and *negative* (direct and critical). Some examples of these comments are shown below:

- Positive comment: `Good starting point; consider hashing passwords (e.g., bcrypt) and adding rate limiting.`
- Neutral comment: `Passwords are compared directly; no hashing or authentication on change_password().`
- Negative comment: `Storing plaintext passwords is unacceptable; remove hardcoded creds and implement hashing.`

We further analyzed feedback comments with VADER [35, 58] to compare feedback styles. For each participant's comment set, VADER outputs four scores: *negative*, *neutral*, *positive*, and *compound* scores. Negative sentiment score was the proportion of text expressing negative emotions; neutral sentiment score was the proportion of text expressing neutral emotions; positive sentiment score was the proportion of text expressing positive emotions; and compound score was the normalized, weighted composite score between -1 and 1 [35]. We calculated the average sentiment score for each country and then (1) cataloged which security issues each participant identified, (2) summarized detection rates by region and seniority, and (3) compared comment tone via the per-engineer VADER scores above.

Development Process Estimation. In addition to technical assessments, participants estimated the time required for common activities in the software lifecycle. These activities were grouped into four categories, reflecting prior work on developer workflows [44, 53]:

- *Issue lifecycle*: begins with investigation and root cause analysis, continues with implementation planning and design, followed by development and testing, then documentation and pull request preparation, and concludes with issue verification and closure.
- *Code review process*: begins with code cleanup and documentation, continues with running tests locally and peer review, then addressing feedback, and concludes with final approval.
- *Deployment process*: begins with build and test automation, continues with staging deployment and testing, then stakeholder review, and concludes with production deployment and post-deployment verification.
- *Bug fix process*: begins with bug investigation and reproduction, continues with fix implementation and testing, then documentation and review, and concludes with deployment and verification.

To analyze these responses, all time estimates were normalized into minutes, ensuring comparability across formats (minutes, hours, days, weeks). We then constructed process flow visualizations, highlighting both the relative time spent per activity and the sequence of tasks. Further analyses identified the top and bottom steps by time consumption, as well as critical path bottlenecks consuming more than 10% of total effort. Cross-country comparisons were finally aggregated into tables, providing insights into where East African and Northwestern European software engineers focus their efforts and how development practices diverge [11]. We present these results in Section 4.

4 Results and Analysis

This section presents the findings of the comparative study of software engineers in East Africa (Rwanda and Uganda) and Northwestern Europe (Sweden and the Netherlands). Results are organized by research question we discussed in Section 3.1: *RQ1: technical skills and problem-solving*, *RQ2: communication and process practices*, and *RQ3: biases and perceptions*. All quantitative analyses are complemented with qualitative interview insights.

4.1 Technical Skills and Problem-Solving (RQ1)

In answer to RQ1, we examine the three tasks introduced in Section 3.3. Below we present the results of each task.

Task 1: Programming Assignment. Table 1 summarizes the solution strategies. Engineers from Sweden and the Netherlands predominantly used efficient hash-based approaches, with some variation toward two-pointer techniques. Rwandan participants mostly relied on nested loops, though a minority also used hash maps. Interviews revealed that several Rwandan participants explicitly recognized more efficient methods, even if they did not implement them. Ugandan engineers showed the highest failure rate, with over half of their submissions nullified.

To further examine coding styles, we used CodeBERT [21] embeddings projected with PCA [1] and clustered with K-means [68] (we considered four clusters, representing four different coding styles). Figures 1 and 2 show clear groupings: engineers from Sweden and the Netherlands cluster tightly, reflecting consistent coding

Table 1: Distribution of solution approaches by country

Country	Hash map/Set	Nested For Loops	Two-Pointer/Counter	Failed/ Other
Rwanda	23%	69%	0%	8%
Uganda	29%	14%	0%	57%
Sweden	80%	10%	10%	0%
Netherlands	55%	27%	18%	0%

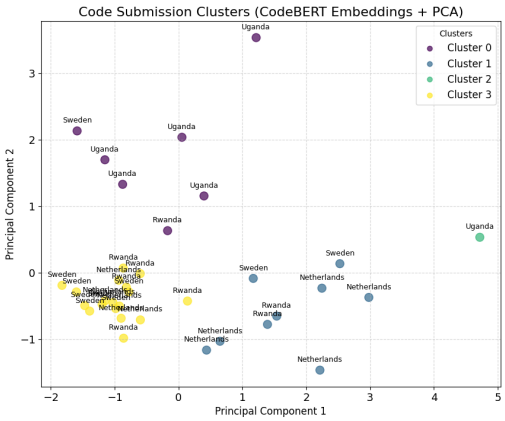

Figure 1: Code submission clusters using CodeBERT embeddings projected via PCA.

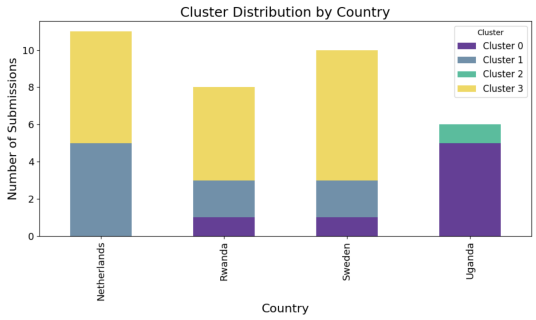

Figure 2: Code submission clusters using CodeBERT embeddings shown through a bar chart.

conventions, while Rwandan engineers overlap substantially with Europeans despite their reliance on nested loops. Ugandan valid submissions diverged, concentrated in alternative clusters. The clusters can be represented in another way using a barchart as shown in Figure 2. Interviews suggested that European consistency reflects standardized curricula, whereas Rwandan variability stems from heterogeneous educational backgrounds and uneven training opportunities.

Task 2: System Design Challenge. The system design task revealed marked regional differences in architectural maturity and variation across experience levels. Engineers from the Netherlands consistently delivered systematic and well-rounded designs, emphasizing security, monitoring, and DevOps practices. Their responses showed little difference between juniors and seniors, reflecting

strong knowledge transfer within teams. Swedish engineers demonstrated clear progression by experience: juniors provided solid foundational designs, while seniors expanded solutions with caching, performance optimization, and service separation.

Rwandan engineers displayed the widest variation. Some gave incomplete or basic answers, while others produced advanced solutions rivaling European seniors, including serverless and document-oriented architectures. Variation was evident across both juniors and seniors, which interviews linked to uneven access to education and professional opportunities. Nonetheless, standout Rwandan engineers showed strong individual potential. Overall, Dutch engineers were the most consistent, Swedish engineers showed the clearest progression with experience, and Rwandan engineers highlighted both challenges and promise. Ugandan submissions were excluded due to integrity concerns and the use of AI tools.

Task 3: Code Review Challenge. Analyzing Task 3 is divided into two parts: Part 1 focuses on security knowledge, and Part 2 involves code review. Here we focus on Part 1, which directly addresses RQ1, and we will discuss Part 2 under RQ2. Security issue identification rates varied considerably across regions and seniority levels. Engineers from Sweden and the Netherlands consistently detected common vulnerabilities such as plaintext password storage and hardcoded credentials. Rwandan seniors, however, outperformed their European peers in breadth of issues identified, while Rwandan juniors were the weakest group overall. Swedish juniors led in awareness, followed by Dutch, with Rwandan juniors trailing. These findings point to a wide senior–junior gap in Rwanda, where seniors rivaled or exceeded European peers, but juniors performed below all groups. For this task, Ugandan submissions were again excluded due to integrity concerns and the use of AI tools.

Figure 3 highlights that across all countries, seniors outperformed juniors, but Rwandan seniors stood out, demonstrating broader coverage of vulnerabilities than Dutch seniors and nearly matching Swedish peers. Interviews suggested that heavier workloads and greater exposure to diverse tasks in Rwanda may accelerate senior competence, while limited access to quality education explains junior underperformance. Rwandan seniors often accumulate more tech-work hours, with one remarking that in Rwanda "work is not only work, but survival" unlike in Europe. Overall, the results show that while European engineers employ more standardized approaches, Rwandan engineers display higher variability: some produced outstanding performances comparable to European standards, while others failed to implement complete solutions. This variability reflects systemic inequalities in education and training opportunities, yet the standout performances illustrate substantial latent potential within East Africa's software engineering workforce.

4.2 Communication and Process Practices (RQ2)

To address this research question, we examined two complementary activities: (1) the sentiment expressed in code review comments (Part 2 of Task 3), which reflects communication styles and feedback practices, and (2) self-reported process time estimates, which provide insight into how engineers allocate effort across different

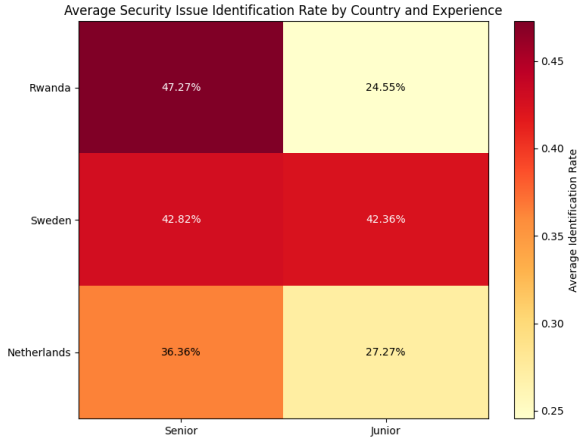

Figure 3: Heatmap comparison of compounded security awareness between senior and junior software engineers across Rwanda, Sweden, and the Netherlands.

Table 2: Sentiment analysis of code review comments.

Country	Negative	Neutral	Positive	Compound
Rwanda	0.049	0.818	0.133	+0.541
Sweden	0.113	0.780	0.107	+0.007
Netherlands	0.153	0.764	0.083	-0.361
Uganda	0.047	0.713	0.240	+0.919

stages of the development lifecycle.

Code Review Sentiment. The code review challenge examined the sentiment of developer comments, with results summarized in Table 2. While neutrality dominated across all countries, the compound sentiment scores highlight clear contrasts. Rwandan and Ugandan engineers provided more positive and supportive feedback (compound scores > 0.5, close to +1), suggesting an emphasis on encouragement. In contrast, Dutch engineers leaned toward more negative tones, while Swedish engineers remained largely neutral.

Interviews revealed that European engineers generally preferred direct critique, which they viewed as constructive and aligned with common norms in their engineering cultures. East African engineers, by contrast, valued supportive tones that reinforced collaborative spirit. These cultural differences have important implications for cross-cultural collaboration: while Rwandan and Ugandan teams may foster more emotionally supportive environments, Swedish and Dutch teams may favor task-oriented directness.

Process Time Estimates. Table 3 shows the self-reported software development process time estimates by country. This table is divided into four process steps, where each is broken down into finer-grained tasks. Participants were asked to estimate how long each task typically takes them to complete. Table 4 shows the steps that take > 10% of the total time. The results reveal clear regional emphases. Swedish and Dutch engineers devoted most of their time to early development stages such as planning, design, and implementation, while Rwandan engineers emphasized deployment activities and post-deployment verification. Ugandan engineers reported much longer durations across nearly all steps. Interviews

suggested these inflated times stemmed largely from English-as-a-second-language challenges rather than genuine inefficiencies. For this reason, Uganda is included in the table for transparency but treated as an outlier in cross-country comparisons.

Comparisons of the fastest and slowest tasks provide further insights. Rwandan engineers were relatively quick at bug investigation, documentation, and deployment, but slower in development and testing. Dutch engineers minimized time on deployment checks and approvals but faced bottlenecks in planning and bug fixing. Swedish engineers completed code review tasks quickly but spent considerable time on build and test automation, planning, and development. Critical path analysis (Table 4) reinforces these contrasts: engineers from Sweden and the Netherlands concentrated their effort on development and testing, while Rwandans invested a larger share in deployment, including automation and post-deployment verification.

Overall, the findings suggest distinct optimization strategies. European engineers seek to reduce rework and technical debt through heavy upfront investment in planning and development, whereas Rwandan engineers focus on robustness by emphasizing collaborative review, deployment, and verification. These differences reflect broader organizational and cultural approaches to balancing efficiency with reliability in software development.

4.3 Biases and Perceptions (RQ3)

Unlike RQ1 and RQ2, which examined measured skills and practices, RQ3 focuses on how software engineers perceive their own and others' competencies. To this end, we analyze the ratings given by participants from each country to all four countries across three skill areas: (1) software development, (2) problem solving, and (3) communication. The perception ratings revealed consistent regional biases.

Rwandan participants generally saw themselves as comparable to Europeans, though they rated Ugandan engineers slightly lower (Figure 4). Many emphasized that Ugandan developers are capable, yet still ranked them beneath Europeans and themselves. They also noted differences between Europeans, describing engineers from Sweden as entrepreneurial and Dutch engineers as hardworking.

Swedish engineers rated themselves the highest, followed by Dutch peers, while placing both Rwandan and Ugandan engineers much lower (Figure 5). They attributed this not to individual ability but to systemic issues such as weaker education and infrastructure in East Africa. Dutch engineers ranked themselves slightly above Swedes and rated East African engineers as much less capable (Figure 6). Interviews revealed that many Dutch participants lacked detailed knowledge about Rwanda or Uganda, often generalizing based on stereotypes of "Africa" rather than direct experience. Ugandan engineers rated themselves slightly higher than Rwandans but still below Europeans (Figure 7), citing superior education and infrastructure in Northwestern Europe. This internalized hierarchy underscores how systemic disparities shape professional confidence.

On average, across all groups, European engineers consistently rated themselves highest and African engineers lowest. Interviews confirmed that participants expected such results: Rwandan team leads anticipated bias against them, while European team leads saw the ratings as reflecting "real" skill distributions. Yet, performance data contradicts these assumptions, as Rwandan seniors often matched or exceeded European peers in technical tasks. These findings reveal a gap between measured ability and perceived competence. While East African engineers expressed relatively balanced views of global capabilities, European engineers systematically underrated their African counterparts. Such perception biases risk undermining equitable collaboration by obscuring complementary strengths and reinforcing global hierarchies within software engineering.

5 Discussion

This section discusses the main findings of the comparative study and their implications for global software development by addressing the three research questions introduced in Section 3.1. Overall, the results indicate that differences between East Africa and Northwestern Europe are shaped less by inherent capability and more by systemic factors, cultural practices, and perception biases.

Technical Skills and Prblem-Solving (RQ1). The technical assessments revealed consistent performance among developers in Northwestern Europe and higher variance in East Africa, particularly in Rwanda. While many Rwandan juniors underperformed, several seniors matched or surpassed European peers, especially in security awareness. These disparities appear to be systemic: European engineers benefited from standardized education and structured career paths, whereas Rwandan engineers described uneven university training and gaps in algorithms or data structures that persisted into their early careers. At the same time, standout East African engineers clustered closely with Europeans in the CodeBERT analysis, especially those with exposure to international consultancies, open-source contributions, or graduate training. Rwandan seniors also reported much longer working hours (often 60+ per week across multiple projects), which may accelerate experiential learning and explain their advanced security knowledge despite weaker formal education. For organizations, this suggests that East African talent should be recognized for its diversity, where some engineers already perform at global standards and others are still developing, with opportunities and exposure playing a key role in shaping outcomes.

Communication and Process Practices (RQ2). Communication styles reflected regional cultural patterns. Sentiment analysis of code reviews showed that Rwandan and Ugandan engineers provided more positive and supportive feedback (compound scores > 0.5), while Dutch engineers leaned toward negative, and Swedish engineers were neutral. Interviews confirmed that Europeans considered direct critique efficient and professional, while East African engineers valued encouragement and group harmony. These align with Hofstede's framework: higher power distance and collectivism in East Africa encourage supportive communication, while lower power distance and individualism in Northwestern Europe favor directness. Process allocations revealed similar contrasts. Dutch and Swedish engineers devoted around 30% of effort to initial design and development, emphasizing strong architecture to minimize rework. Rwandan engineers focused more on deployment, verification, and

Table 3: Self-reported software development process time (minutes) estimates by country.

Process	Step	RW	NL	SE	UG
Issue Lifecycle	Development/testing	261.1	720.0	792.0	1920.0
	Documentation/PR	101.1	96.0	109.0	960.0
	Planning/design	194.4	189.0	264.0	1600.0
	Investigation/analysis	165.9	111.0	132.0	1240.0
	Verification/closure	141.7	51.0	88.0	840.0
Code Review Process	Addressing feedback	143.9	64.5	90.0	1380.0
	Cleanup/documentation	236.1	58.5	76.5	1140.0
	Final approval	82.2	34.5	51.0	1190.0
	Peer review process	87.2	96.0	93.0	1360.0
	Running tests locally	132.4	57.0	57.5	1170.0
Deployment Process	Build/test automation	287.8	135.0	264.0	1145.0
	Post-deploy verification	335.6	51.0	58.0	1400.0
	Production deployment	126.7	54.0	72.0	1120.0
	Staging deploy/testing	12.0	64.5	70.0	1080.0
	Stakeholder review	124.4	61.5	76.5	1360.0
Bug Fix Process	Investigation/reproduction	60.0	147.0	213.0	1400.0
	Deployment verification	45.6	78.0	96.0	980.0
	Documentation/review	25.6	52.5	54.0	1080.0
	Implementation/testing	46.1	261.0	261.0	1120.0

Note: Values represent average time in minutes. RW=Rwanda, NL=Netherlands, SE=Sweden, UG=Uganda.

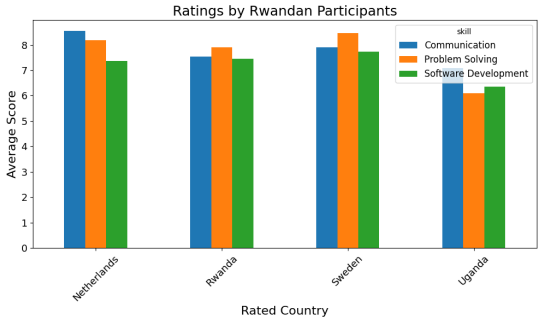

Figure 4: Perception-based given by Rwandan participants to developers from Rwanda, Sweden, the Netherlands, and Uganda.

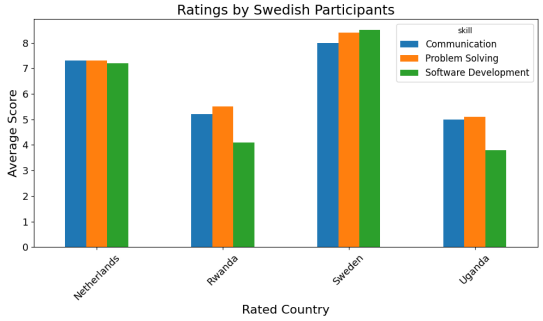

Figure 5: Perception-based ratings given by Swedish participants to software engineers from Rwanda, Sweden, the Netherlands, and Uganda.

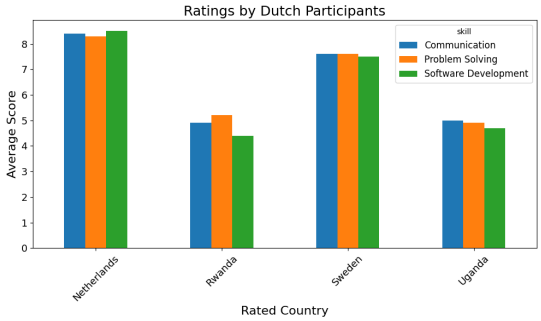

Figure 6: Perception-based ratings given by Dutch participants to software engineers from Rwanda, Sweden, the Netherlands, and Uganda.

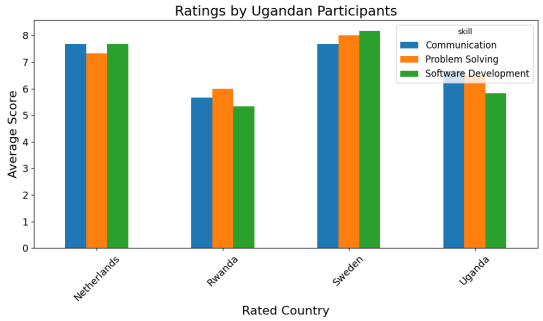

Figure 7: Perception-based ratings given by Ugandan participants to software engineers from Rwanda, Sweden, the Netherlands, and Uganda.

review, often citing rigorous DevOps pipelines and collective bug fixing. Rather than inefficiency, this reflects different optimization strategies: Northwestern Europe teams invest upfront, while East African teams distribute risk across collaborative verification. In global teams, these strategies could be complementary, combining robustness in design with resilience in testing and deployment.

Biases and Perceptions (RQ3). Perception studies revealed gaps between measured skills and perceived ability. European developers consistently rated themselves and one another higher, often

Table 4: Critical path analysis: steps taking >10% of total time in minutes

Country	Process Step	Time	%
RW	Dev & testing (Issue Lifecycle)	261.1	10.0%
	Post-deploy verification (Deployment)	335.6	12.9%
	Build & test automation (Deployment)	287.8	11.0%
NL	Dev & testing (Issue Lifecycle)	720.0	30.2%
	Fix impl & testing (Bug Fix)	261.0	11.0%
SWE	Dev & testing (Issue Lifecycle)	792.0	27.1%

scoring East African peers below 5/10, despite evidence of individual excellence among Rwandan seniors. Many admitted to having limited knowledge of East Africa's ecosystems, instead generalizing from stereotypes of weak infrastructure and education. By contrast, Rwandans tended to view themselves as equal to Europeans but superior to Ugandans, while Ugandans rated Europeans highest, reflecting internalized hierarchies shaped by systemic inequities. These perceptions risk undermining confidence and collaboration: Rwandan seniors who matched European performance still expected to be underestimated. Addressing such biases is as crucial for global software development as addressing technical gaps, since they can erode trust and prevent recognition of complementary strengths.

Methodological Reflections. The pragmatic mixed-methods design enabled triangulation across programming tasks, design challenges, process estimates, and interviews. Strengths include bootstrap resampling that confirmed the stability of clustering results and high inter-rater reliability in perception ratings. However, the sample was small (10–15 per country) and recruited via professional networks, skewing toward high achievers in Europe and Kigali-based consultancies. Ugandan data was further limited by language barriers and unauthorized AI use. Findings should therefore be interpreted as exploratory rather than definitive. Future research should employ larger, possibly longitudinal, samples to track skill evolution over time, incorporate more diverse East Africa contexts beyond capital cities, and complement self-reports with log data for higher precision. Nevertheless, this study shows the value of pragmatic, flexible methodologies in capturing both quantitative patterns and contextual nuance in cross-cultural software engineering.

Implications. Taken together, the findings demonstrate that global collaboration is shaped not only by technical ability but also by systemic inequities, cultural practices, and perception hierarchies. With respect to **RQ1**, East African engineers represent a high-variance talent pool where standout performers reach global standards despite systemic disadvantages. For **RQ2**, communication and process practices reflect culturally adaptive optimization strategies that could be complementary in cross-regional teams. Finally, **RQ3** shows that perception biases, both external stereotypes and internalized hierarchies, pose persistent risks for equitable collaboration. Recognizing and addressing these dynamics is critical for building effective and inclusive global software development teams.

6 Conclusion

This study compared software engineering practices in East Africa (Rwanda and Uganda) and Northwestern Europe (Sweden and the Netherlands), addressing three research questions on technical competence, communication practices, and perception biases in global software development. The findings show that observed differences across regions are better explained by systemic and cultural factors than by inherent technical ability. European engineers demonstrated strong consistency in coding and system design, whereas Rwandan engineers showed greater performance variation, with several senior engineers performing at levels comparable to or exceeding those of their European peers, particularly in security-related tasks. These results challenge assumptions that outsourcing to Africa entails a quality trade-off and instead point to complementary strengths: Northwestern European engineers tend to emphasize upfront design and standardization, whereas East African engineers place greater emphasis on collaborative verification, rigorous DevOps practices, and supportive communication. Despite these complementary capabilities, significant perception gaps persist. European participants systematically underestimated the competence of their East African counterparts, even when objective measures showed comparable or superior performance. This mismatch between measured performance and perceived competence highlights how everyday evaluation practices can reproduce epistemic unfairness and reinforce existing inequalities in global software labor.

Methodologically, the pragmatic mixed-methods design proved valuable for linking quantitative performance with qualitative cultural insights; however, limitations include a small sample size, selection bias, and inconsistent data quality in Uganda. Future research should expand to larger, more diverse samples, include longitudinal tracking of junior progression, and complement self-reports with repository and process log analysis. Developing sentiment analysis tools tailored for technical communication across cultures would further strengthen validity. Overall, this work provides one of the first empirical baselines of cross-cultural software engineering in East Africa. By documenting both technical and cultural dimensions, it contributes to a more nuanced understanding of global collaboration, highlighting the untapped potential of African software engineers when systemic barriers are addressed. The findings underscore that effective global software development depends not only on technical alignment but also on bridging cultural practices and perception gaps.

Generative AI Usage Statement

We used generative AI tools to help polish the text and improve clarity.

Acknowledgment

We thank InspireLab at KTH Royal Institute of Technology for partially funding this project.

References

[1] Hervé Abdi and Lynne J Williams. 2010. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics* 2, 4 (2010), 433–459.

[2] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified pre-training for program understanding and generation. *arXiv preprint arXiv:2103.06333* (2021).

[3] Toufique Ahmed, Amiangshu Bosu, Anindya Iqbal, and Shahram Rahimi. 2017. SentiCR: A customized sentiment analysis tool for code review interactions. In

2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 106–111.

[4] Victor Oluwatosin Ajayi. 2023. A review on primary sources of data and secondary sources of data.

[5] Abdulaziz Alsahli, Hameed Khan, and Sultan Alyahya. 2017. Agile development overcomes GSD challenges: a systematic literature review. *International Journal of Computer Science and Software Engineering* 6, 1 (2017), 7.

[6] Hala Assal, Srivathsan G. Morkonda, Muhammad Zaid Arif, and Sonia Chxiasson. 2025. Software security in practice: knowledge and motivation. *Journal of Cybersecurity* (2025). doi:10.1093/cybsec/tyaf005

[7] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*. 1–2. doi:10.1109/ICSE.2013.6606647 Foundational study on modern code review practices, covering goals like quality improvement, finding defects, knowledge transfer, team awareness, and associated challenges..

[8] Engineer Bainomugisha, R. Hebig, and Michel R. V. Chaudron. 2021. Emerging Software Engineering Research Networks in (East) Africa. *ACM SIGSOFT Software Engineering Notes* 46 (2021), 18–22. doi:10.1145/3448992.3448996

[9] Anne E Betzner. 2008. *Pragmatic and dialectic mixed method approaches: An empirical comparison.* University of Minnesota.

[10] Fabio Calefato, Filippo Lanubile, Federico Maiorano, and Nicole Novielli. 2018. Sentiment polarity detection for software development. In *Proceedings of the 40th International Conference on Software Engineering*. 128–128.

[11] Min Chen and Amos Golan. 2015. What may visualization processes optimize? *IEEE transactions on visualization and computer graphics* 22, 12 (2015), 2619–2632.

[12] Emma Clarke and John Visser. 2019. Pragmatic research methodology in education: possibilities and pitfalls. *International Journal of Research & Method in Education* 42, 5 (2019), 455–469.

[13] Ricardo Colomo-Palacios, Edmundo Tovar-Caro, and Ángel García-Crespo. 2018. *Identifying Technical Competences of IT Professionals: The Case of Software Engineers.* Technical Report. Universidad Carlos III de Madrid / Universidad Politécnica de Madrid. Also known in Spanish as "Niveles competenciales para los perfiles de Ingenieros de Software"..

[14] John W Creswell, Ann Carroll Klassen, Vicki L Plano Clark, Katherine Clegg Smith, et al. 2011. Best practices for mixed methods research in the health sciences. *Bethesda (Maryland): National Institutes of Health* 2013 (2011), 541–545.

[15] Ella Dagan, Anita Sarma, Alison Chang, Sarah D'Angelo, Jill Dicker, and Emerson Murphy-Hill. 2023. Building and sustaining ethnically, racially, and gender diverse software engineering teams: A study at google. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 631–643.

[16] Advait Deshpande, Helen Sharp, Leonor Barroca, and Peggy Gregory. 2016. Remote working and collaboration in agile teams. (2016).

[17] Marie Devlin and Chris Phillips. 2010. Assessing competency in undergraduate Software Engineering teams. In *IEEE EDUCON 2010 Conference*. IEEE, 271–278.

[18] Jan Drahokoupil. 2015. *The Outsourcing Challenge: Organizing Workers Across Fragmented Production Networks.* European Trade Union Institute (ETUI). https://www.etui.org/Publications2/Books/The-outsourcing-challenge-Organizing-workers-across-fragmented-production-networks

[19] Christof Ebert, Marco Kuhrmann, and Rafael Prikladnicki. 2016. Global Software Engineering: Evolution and Trends. *IEEE 11th International Conference on Global Software Engineering* (2016). doi:10.1109/ICGSE.2016.19

[20] Christof Ebert, Marco Kuhrmann, and Rafael Prikladnicki. 2016. Global Software Engineering: Evolution and Trends. *IEEE 11th International Conference on Global Software Engineering* (2016). doi:10.1109/ICGSE.2016.19

[21] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *arXiv preprint arXiv:2002.08155* (2020).

[22] Paul Feyerabend. 2020. *Against method: Outline of an anarchistic theory of knowledge.* Verso Books.

[23] RM Florea. 2023. The Software Tester: An Exploration of the Skills and Practice of the Role. *University of Oslo* (2023). https://www.duo.uio.no/bitstream/handle/10852/101865/1/PhD-Florea-2023.pdf

[24] V. Garousi, G. Giray, E. Tüzün, and C. Catal. 2019. Aligning software engineering education with industrial needs: A meta-analysis. *Journal of Systems and Software* (2019). https://pure.qub.ac.uk/files/199651164/JSS_Aligning_SE_education_June_10.pdf

[25] C. A. Kopati Gbali. 2021. Impact of National Policies on Knowledge Transfer to Small Tech Companies: The Case of Rwanda. *GRIPS Discussion Paper* (2021), 4–5. https://grips.repo.nii.ac.jp/record/2000139/files/k295_report_phd20605.pdf

[26] Dhashen Govender. 2024. *The Rise of African Tech Talent in Global Software Outsourcing.* Scrums.com. https://scrums.com/software-outsourcing/african-tech-talent-global-software-outsourcing Accessed: May 2025.

[27] Greg Guest, Arwen Bunce, and Laura Johnson. 2006. How many interviews are enough? An experiment with data saturation and variability. *Field methods* 18, 1 (2006), 59–82.

[28] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. 2020. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366* (2020).

[29] Marcus Hägglund, Francisco J Pena, Sepideh Pashami, Ahmad Al-Shishtawy, and Amir H Payberah. 2021. Coclubert: Clustering machine learning source code. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 151–158.

[30] Payam Hanafizadeh and Ahad Zareravasan. 2020. A systematic literature review on IT outsourcing decision and future research directions. *Journal of Global Information Management (JGIM)* 28, 2 (2020), 160–201.

[31] Monique Hennink and Bonnie N Kaiser. 2022. Sample sizes for saturation in qualitative research: A systematic review of empirical tests. *Social Science & Medicine* 292 (2022), 2,6. doi:10.1016/j.socscimed.2021.114523

[32] Luiz Hespanhol, Caio Sain Vallio, Lucíola Menezes Costa, and Bruno T Saragiotto. 2019. Understanding and interpreting confidence and credible intervals around effect estimates. *Brazilian journal of physical therapy* 23, 4 (2019), 290–301.

[33] Geert Hofstede. 2011. Dimensionalizing Cultures: The Hofstede Model in Context. *Online Readings in Psychology and Culture* 2, 1 (December 2011), 8. doi:10.9707/2307-0919.1014

[34] Bret Hunley, Smarajit Chakraborty, and Stephen MacDonald. 2018. The impact of cultural communication on team performance. (2018).

[35] Clayton Hutto and Eric Gilbert. 2014. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the international AAAI conference on web and social media*, Vol. 8. 216–225.

[36] Zafar Iqbal and Aasim Munir Dad. 2013. Outsourcing: A review of trends, winners & losers and future directions. *International Journal of Business and Social Science* 4, 8 (2013).

[37] Mwangi Karanja, Lawrence Xavier Thuku, and John Patrick Kangethe. 2015. Software Development Industry in East Africa: Knowledge Management Perspective and Value Proposition. *Africa Casebook - Synergies in African Business and Management Practices* (2015), 26–37. https://www.researchgate.net/publication/274730121_Software_Development_Industry_In_East_Africa_Knowledge_Management_Perspective_And_Value_Proposition

[38] Alison Karasz and Theodore M Singelis. 2009. Qualitative and mixed methods research in cross-cultural psychology. 909–916 pages.

[39] Arif Ali Khan, Mohammad Shameem, Mohammad Nadeem, and Muhammad Azeem Akbar. 2021. Agile trends in Chinese global software development industry: Fuzzy AHP based conceptual mapping. *Applied Soft Computing Journal* 102 (2021). doi:10.1016/j.asoc.2021.107090 Accessed: 2025-02-28.

[40] S. Krishna, Sundeep Sahay, and Geoff Walsham. 2004. Managing Cross-Cultural Issues in Global Software Outsourcing. *Commun. ACM* 47, 4 (April 2004), 62–66.

[41] John Law. 2004. *After method: Mess in social science research.* Routledge.

[42] LeetCode. [n. d.]. Two Sum. https://leetcode.com/problems/two-sum/. Accessed: 2025-05-05.

[43] Per Lenberg, Robert Feldt, Lucas Gren, Lars Göran Wallgren Tengberg, Inga Tidefors, and Daniel Graziotin. 2024. Qualitative software engineering research: Reflections and guidelines. *Journal of Software: Evolution and Process* 36, 6 (2024), e2607.

[44] Sherlock A Licorish and Stephen G MacDonell. 2017. Exploring software developers' work practices: Task differences, participation, engagement, and speed of task resolution. *Information & Management* 54, 3 (2017), 364–382.

[45] Tanishq Likhi. 2022. Effects of the Power Distance Index on Multicultural Software Engineering Teams. (2022), 19–25. Discussion on unconscious biases in work distribution, communication, and cultural adaptation..

[46] Eve MacGregor, Yvonne Hsieh, and Philippe Kruchten. 2005. The impact of intercultural factors on global software development. In *Canadian Conference on Electrical and Computer Engineering, 2005.* IEEE, 920–926.

[47] Wustari Mangundjaya, Imo Gandakusuma, and Mira Sekar Arumi. 2023. POWER DISTANCE, UNCERTAINTY AVOIDANCE AND COMMITMENT TO CHANGE. *Jurnal Scientia* 12, 03 (2023), 2852–2860.

[48] Marcelo Marinho, Alexandre Luna, and Sarah Beecham. 2018. Global software development: practices for cultural differences. 299–317 pages.

[49] Jorge Martinez-Gil. 2024. Augmenting the Interpretability of GraphCodeBERT for Code Similarity Tasks. *arXiv preprint arXiv:2410.05275* (2024). arXiv:2410.05275 [cs.IR]

[50] S. Matthiesen and P. Bjørn. 2023. Implicit bias and negative stereotyping in global software development and why it is time to move on! *Journal of Software: Evolution and Process* 35, 2 (2023), e2435. doi:10.1002/smr.2435

[51] Josetta S McLaughlin, Gerald W McLaughlin, and John A Muffo. 2001. Using qualitative and quantitative methods for complementary purposes: A case study. *New directions for institutional research* 2001, 112 (2001), 15–44.

[52] Fernando Merino. 2017. Offshoring, outsourcing and the economic geography of Europe. *Papers in Regional Science* 96, 2 (2017), 299–324.

[53] André N Meyer, Earl T Barr, Christian Bird, and Thomas Zimmermann. 2019. Today was a good day: The daily life of software developers. *IEEE Transactions on Software Engineering* 47, 5 (2019), 863–880.

[54] Ministry of ICT and National Guidance. 2024. *MoICT eMagazine.* Government Publication Issue 001. Ministry of ICT and National Guidance, Uganda. https://ict.go.ug/site/documents/MoICT_eMag_Issue001_2024.pdf Accessed: 13 May 2023.

[55] L. Mundere and P. Marešová. 2016. Potential of ICT Industry for Economic Growth in Developing Countries - Case Study of Rwanda. *Advanced Science Letters* (2016), 48–50, 56. https://theses.cz/id/wccx2k/STAG84980.pdf

[56] Karanja Evanson Mwangi, Lawrence Xavier Thuku, and John Patrick Kangethe. 2015. Software development industry in East Africa: Knowledge management perspective and value proposition.

[57] Norrsken Foundation. 2025. Norrsken House Kigali. https://www.norrsken.org/houses/kigali Accessed: 2025-01-27.

[58] Martin Obaidi and Jil Klünder. 2021. Development and Application of Sentiment Analysis Tools in Software Engineering: A Systematic Literature Review. *Evaluation and Assessment in Software Engineering (EASE 2021)* (2021). doi:10.1145/3463274.3463328

[59] Akani Prince Oluleye and Adebiyi Adeniyi Mayowa. 2025. Cross-Cultural Dynamics and Teamwork Effectiveness in the Construction Industry: A Review Exploratory Study of Professional Interactions in Nigeria. *Researchers Journal of Science and Technology* 5, 3 (2025), 41–57.

[60] Chuks Otioma. 2019. Exploring the pattern and framework conditions of technology-based entrepreneurial activities in Africa. In *African Economic Conference 2019: Jobs, Entrepreneurship and Capacity Development for African Youths.* African Development Bank, Sharm El Sheikh, Egypt. https://aec.afdb.org/sites/default/files/papers/362-otioma_chuks-exploring_the_pattern_and_framework_conditions_of_technology-based_entrepreneurial_activities_in_africa.pdf Accessed: May 13, 2025.

[61] J. Palokangas. 2013. Agile around the world: How agile values are interpreted in national cultures. *Master's Thesis* (2013). https://core.ac.uk/download/pdf/250133209.pdf

[62] Massimiliano Di Penta and Damian Andrew Tamburri. 2017. Combining Quantitative and Qualitative Studies in Empirical Software Engineering Research. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C).* 499–500. doi:10.1109/ICSE-C.2017.163

[63] Albert Sahakyan and Tigran Sloyan. 2023. *General Coding Skills Evaluation Framework: Technical Research Paper.* Technical Report. CodeSignal. https://discover.codesignal.com/rs/659-AFH-023/images/General-Coding-Framework-Technical-Research-Paper-CodeSignal.pdf Originally published July 2019; Updated April 2023.

[64] Sundeep Sahay, Brian Nicholson, and Srinivas Krishna. 2003. *Global IT outsourcing: software development across borders.* Cambridge University Press.

[65] Emmanuel Eturpa Salami, Clement Onyemaechi Nwabuokei, and Kayode Ekundayo. 2024. Software Development and the Security Challenges in Developing Countries. *Global Scientific Journal* 12, 12 (2024), 927–941. https://www.globalscientificjournal.com

[66] Nazish Saleem, Sanjay Mathrani, and Nazim Taskin. 2019. Understanding the Different Levels of Challenges in Global Software Development. *IEEE Xplore* (2019). doi:10.1109/ICGSE.2019.8807743

[67] Philip Sedgwick. 2013. Convenience sampling. *Bmj* 347 (2013).

[68] Kristina P Sinaga and Miin-Shen Yang. 2020. Unsupervised K-means clustering algorithm. *IEEE access* 8 (2020), 80716–80727.

[69] R. Sitas, L. Cirolia, A. Pollio, and A. G. Sebarenzi. 2022. Platform Politics and Silicon Savannahs: The Rise of On-Demand Logistics and Mobility in Nairobi and Kigali. *International Centre for Cities* (2022). https://www.researchgate.net/profile/Andrea-Pollio/publication/360773802_Platform_Politics_and_Silicon_Savannahs_The_rise_of_on-demand_logistics_and_mobility_in_Nairobi_and_Kigali/links/6289fae239fa21703165ccf4/Platform-Politics-and-Silicon-Savannahs-The-rise-of-on-demand-logistics-and-mobility-in-Nairobi-and-Kigali.pdf

[70] Günter K Stahl, Martha L Maznevski, Andreas Voigt, and Karsten Jonsen. 2010. Unraveling the effects of cultural diversity in teams: A meta-analysis of research on multicultural work groups. *Journal of international business studies* 41 (2010), 690–709.

[71] Margaret-Anne Storey, Rashina Hoda, Alessandra Maciel Paz Milani, and Maria Teresa Baldassarre. 2024. Guiding Principles for Mixed Methods Research in Software Engineering. *arXiv preprint arXiv:2404.06011* (2024), 4,6–7.

[72] Taarifa Rwanda. 2023. *Rwanda Launches '1 Million Rwandan Coders' to Build Africa's Digital Workforce.* Taarifa Rwanda. https://taarifa.rw/rwanda-launches-1-million-rwandan-coders-to-build-africas-digital-workforce/ Accessed: May 2025.

[73] Maureen Tanner. 2009. Communication and Culture in Global Software Development: The Case of Mauritius and South Africa. *Journal of Information, Information Technology & Organizations* 4 (2009).

[74] Maureen Tanner. 2010. Software Methods in Global Software Development: the Case of Mauritius and South Africa. In *International Conference on Information Management and Evaluation.* Academic Conferences International Limited, 388.

[75] TechGyant. 2023. *ICT Powers Rwanda's GDP with 3.5% All-Time High Growth in 2023.* TechGyant. https://techgyant.com/ict-powers-rwandas-gdp-with-35-all-time-high-growth-in-2023/ Accessed: 13 May 2025.

[76] Author(s) Unknown. 2023. The role of intercultural differences and challenges faced in negotiating active mine sites' rehabilitation objectives from Africa to Europe. *Journal of Environmental Management* (2023), 2–4, 16. https://www.sciencedirect.com/science/article/pii/S2214790X2300151X Accessed: 2025-02-28.

[77] Zhiyuan Wan, Yun Zhang, Xin Xia, Yi Jiang, and David Lo. 2023. Software architecture in practice: Challenges and opportunities. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* 1457–1469.

[78] Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi DQ Bui, Junnan Li, and Steven CH Hoi. 2023. Codet5+: Open code large language models for code understanding and generation. *arXiv preprint arXiv:2305.07922* (2023).

[79] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859* (2021).

[80] Y Wang, Y Yue, and G Zhang. 2024. Uncovering non-native speakers' experiences in global software development teams: A bourdieusian perspective. *Computer Supported Cooperative Work (CSCW)* (2024), 1–3. https://link.springer.com/article/10.1007/s10606-024-09504-y

[81] Daniel Welsch, Luisa Burk, David Mötefindt, and Michael Neumann. 2023. Navigating cultural diversity: barriers and potentials in multicultural agile software development teams. *arXiv preprint arXiv:2311.12061* (2023).

[82] Islam Zada and Sara Shahzad. 2015. Issues and implications of scrum on global software development. *Bahria University Journal of Information & Communication Technologies (BUJICT)* 8, 1 (2015).

[83] Eoin Ó Conchúir, Pär J. Ågerfalk, Helena H. Olsson, and Brian Fitzgerald. 2009. Global software development: Where are the benefits? *Commun. ACM* 52, 8 (2009), 127–131. doi:10.1145/1536616.1536648