Degree Project in Computer Science and Engineering

Second cycle, 30 credits

A Comparative Study of Software Engineers in East Africa and North Western Europe Based on Skills, Communication, Organizational Culture, and Perceptions

A Pragmatic Mixed-Methods Approach to Understanding Global Software Development Dynamics Between Rwanda, Uganda, Sweden, and the Netherlands

SAM KHOSRAVI

A Comparative Study of Software Engineers in East Africa and North Western Europe Based on Skills, Communication, Organizational Culture, and Perceptions

A Pragmatic Mixed-Methods Approach to Understanding Global Software Development Dynamics Between Rwanda, Uganda, Sweden, and the Netherlands

SAM KHOSRAVI

Master's Programme, Computer Science, 120 credits
Date: July 2, 2025

Supervisors: Amir H. Payerah, Faiza A. Bukhsh
Examiner: Olov Engwall

Swedish title: En jämförande studie av mjukvaruingenjörer i Östafrika och Nordvästeuropa baserat på färdigheter, kommunikation, organisationskultur och uppfattningar
Swedish subtitle: En pragmatisk mixed-methods strategi för att förstå global mjukvaruutveckling mellan Rwanda, Uganda, Sverige och Nederländerna

© 2025 Sam Khosravi

Abstract

This research examines cross-cultural dynamics in global software development by comparing software engineers from East Africa (Rwanda and Uganda) and North Western Europe (Sweden and the Netherlands). Despite growing African tech hubs, empirical research looking into the technical competencies, communication styles, and organizational practices of East African software engineers remain limited, despite potential time-zone alignment advantages with European partners. Most existing studies on global software development focus on established outsourcing destinations in Asia, Eastern Europe, and South America.

This thesis addresses this research gap by using a pragmatic mixed-methods approach, combining quantitative and qualitative data, which was collected through standardized programming challenges, system design tasks, code reviews, and in-depth interviews with 48 software engineers across the four countries. Technical solutions were analyzed using Machine Learning techniques, including CodeBERT embeddings and clustering analysis, while communication styles were evaluated through sentiment analysis of code reviews using Valence Aware Dictionary and Sentiment Reasoner from the Natural Language Toolkit.

The results reveal insights challenging conventional assumptions about global software engineering. Rwandan senior software engineers showed technical skills comparable to their European counterparts, while Rwandan junior software engineers were outclassed by their European counterparts. Communication styles differed across regions, with East African software engineers providing more positive and supportive feedback (compound sentiment scores >0.5) compared to the more critical, direct approach of European software engineers (compound score -0.361 for Dutch software engineers). Process priorities also varied as European teams allocated more time to initial development and design, while East African teams put more focus on code reviews and deployment processes. Most significantly, the study found perception gaps with European software engineers consistently underrating the capabilities of East Africans despite measured performance showing individual excellence and overlap between regions in terms of skills. These findings provide empirical evidence that can inform more equitable global software development practices and enhance cross-cultural collaboration by leveraging the complementary strengths adherent in each region. Furthermore it challenges biases that may limit opportunities for talented software engineers from emerging regions, and contributes to a more inclusive understanding of global software engineering dynamics in previously understudied contexts.

Keywords

Global Software Development, Cross-cultural Collaboration, Software Engineering Skills, East Africa, North Western Europe, Perception Bias

Sammanfattning

Denna studie undersöker tvärkulturell dynamik inom global mjukvaruutveckling genom att jämföra mjukvaruingenjörer från Östafrika mot de från Nordvästa Europa. I studien ingår Rwanda och Uganda som skall representera Östafrika, medan Sverige and Nederländerna representerar Nordvästra Europa. Trots att Östafrikanska länder utvecklat både infrastruktur och teknisk kompetens de senaste åren saknas det empirisk forskning om östafrikanska utvecklares tekniska kompetenser, kommunikationssätt och organisatoriska metoder, särskilt i jämförelse med andra mer etablerade outsourcing destinationer i Asien och Östeuropa.

Studien omfattar 48 utvecklare, och använder en pragmatisk mixed-methods ansats som kombinerar kvantitativ och kvalitativ data. Datainsamlingen omfattade standardiserade programmeringsutmaningar, systemdesign, kodgranskningar och djupintervjuer. Maskininlärningsmetoder såsom CodeBERT-embeddings och klusteranalys användes för att analysera tekniska lösningar, medan kommunikationsstilar utvärderades genom sentimentanalys med VADER från NLTK.

Resultaten visar att seniora utvecklare från Rwanda besitter tekniska färdigheter som är jämförbara med deras europeiska motsvarigheter, däremot presterar juniora utvecklare från Rwanda sämre. Kommunikationsstilar skiljer sig markant mellan de två regionerna, då östafrikanska utvecklare gav mer positiv feedback på kodgranskningar, jämfört med européers mer kritiska och direkta tillvägagångssätt. Processprioriteringar varierade också, där europeiska team fokuserade mer på initial utveckling medan östafrikanska team fokuserade mer på kodgranskning och driftsättning. Mest anmärkningsvärt var de perceptions skillnader som hittades, där europeiska utvecklare konsekvent underskattade östafrikanska utvecklares förmågor trots uppmätta prestationer som visade individuell förmåga och överlapp mellan regionerna. Dessa fynd ger empiriska bevis som kan bidra till mer rättvisa globala mjukvaruutvecklings möjlighter och utmana fördomar som begränsar möjligheter för talangfulla utvecklare från tillväxtregioner.

Nyckelord

Global Mjukvaruutveckling, Tvärkulturellt Samarbete, Mjukvaruingenjörsfärdigheter, Östafrika, Nordvästeuropa, Uppfattningsbias

Acknowledgments

I would like to express my sincere gratitude to my supervisor Amir H. Payberah for his guidance, continuous support, and insightful feedback throughout this research project. I would also like to thank Faiza Bukhsh for her valuable guidance during the early stages of my thesis, particularly in exploring process mining and visualization, as well as facilitating a visit to the University of Twente. A further thanks to Amir for being a role model to all KTH students and putting so much effort into work related to social justice, and for helping me come up with my thesis topic.

I would like to give a large thank you to Nicole Van Helst for giving me the original impetus for the thesis, and helping shape it as well. A further thanks for putting me in contact with various software engineers, companies and universities from Rwanda and the Netherlands. Further I would like to thank Yannick Kabayiza and Brian Gharibaan for hosting me at their offices in Kigali, as well as Brian letting me stay at his place of residence in Rwanda. I would also like to thank the whole team at Awesomity Labs for their hospitality and friendliness, where I also have found lifelong friends. A thank you to the software engineers from Code Of Africa as well for participating in the study.

In Uganda, I want to thank Sseruwagi Abdallah for putting me in contact with many software engineers. I also want to thank The Ministry of ICT and National Guidance for hosting me, as well as everyone from STEM Center Kampala and The Innovation Village in Uganda.

This research would not have been possible without the cooperation of the software engineers from Rwanda, Uganda, Sweden, and the Netherlands who generously shared their time, experiences, and insights. Their contributions are the backbone of this study, and I am very thankful towards them. Furthermore, I want to thank all my former and current colleagues and classmates for volunteering to join my sometimes 2 hours long interviews.

Finally, my deepest thanks go to my family and friends for their unwavering support, patience, and encouragement throughout my academic journey. Their belief in me has been a constant source of motivation.

Stockholm, July 2025
Sam Khosravi

Contents

List of Figures

List of Tables

List of acronyms and abbreviations

2FA	Two-Factor Authentication
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CI	Confidence Interval
CI/CD	Continuous Integration/Continuous Deployment
codeBERT	Code Bidirectional Encoder Representations from Transformers
DevOps	Development and Operations
EA	East Africa
GSD	Global Software Development
ICT	Information and Communications Technology
ML	Machine Learning
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NWE	North Western Europe
PCA	Principal Component Analysis
RQ	Research Question
SDG	Sustainable Development Goals
SWEBOK	Software Engineering Body of Knowledge
VADER	Valence Aware Dictionary and Sentiment Reasoner

Chapter 1

Introduction

The initial impetus for this study came from a conversation with a member of the Rwandan delegation in the Netherlands, where concerns were raised about the perception of East African software engineers, particularly those from Rwanda, as being less capable. This perception is not only a local issue the Rwandans face but it is reflecting a broader challenge in Global Software Development (GSD) where stereotypes and biases can influence trust in international collaborations, as well as influence career opportunities for developers from emerging countries.

However, this concern is too narrow to provide insight into a broader engineering context. Instead the scope is broadened to a comparative analysis of software engineers from countries at different stages of development to be able to address both a pressing industrial need and enhance academic insights into cross-cultural software development. This study focuses on North Western Europe (NWE) and East Africa (EA), specifically comparing Sweden and the Netherlands in NWE with Rwanda and Uganda in EA.

This study not only addresses a pressing industrial need, but also aims to contribute to the academic understanding of cross-cultural software development practices. The geographical focus is particularly relevant as the time-zone alignment between NWE and EA offers outsourcing opportunities and real-time collaboration, making these regions a valuable case study for exploring GSD dynamics.

The primary objective of this thesis is to compare the software engineers across the following four dimensions:

- **Technical Skills -** How do coding skills and problem-solving approaches differ across regions?
- **Communication Styles -** What are the differences in communication

patterns, including sentiment and cultural context?

- **Organizational Culture -** How do organizational practices and work environments shape software development workflows?
- **Biases and Perception -** What biases exists, and how could these influence cross-cultural collaboration and perceptions of skills?

Throughout the study, the terms software engineer, software developer, engineer, and developer are used interchangeably. All refer to individuals who write code and are responsible for software engineering tasks.

1.1 Research Questions

Previous studies, as highlighted in Section 2.3, have used qualitative methods primarily to examine cross-cultural communication styles and organizational norms, while this study adopts a mixed-method approach to provide a more comprehensive analysis of cultural influences on software engineering practices. By integrating both quantitative assessments (standardized coding tests) and qualitative evaluations (in depth interviews), this study captures both objective metrics and contextual insights.

This methodological choice is directly related to the Research Question (RQ), which are as follows:

- RQ1: How do software engineers in East Africa and North Western Europe differ in coding skills and problem-solving strategies?
- RQ2: How do communication styles and organizational practices differ between regions?
- RQ3: What biases exist across the countries and how would this affect collaboration between software engineers from East Africa and North Western Europe?

As the study combines quantitative and qualitative methods, it offers a holistic evaluation of technical competencies, communication styles, and organizational practices. This approach advances existing research by providing empirical data from underrepresented regions such as Rwanda and Uganda, with the aim of contributing new insights to the discourse on GSD and cross-cultural collaboration. Such a comparison cross regions including each two countries from EA and NWE has not been made before.

1.2 Problem

Historically, Africa has often been overlooked as an outsourcing destination in GSD due to perceptions of inadequate infrastructure [1], and in turn, competence. Although such challenges persist in some areas, recent investments, particularly in Rwanda, are transforming some countries in EA into tech hubs [2]. Despite these advancements and focused efforts to foster Information and Communications Technology (ICT) talent, most existing studies concentrate on outsourcing to Asia or Eastern Europe [3–6], while empirical investigations into the experiences of software engineers in EA remain scarce. When African countries are studied, it is often more developed countries such as Kenya and South Africa [7–10], and the consideration of Rwandas immense national investments in fostering IT talent is not taken into account, one such example being their dedication to train one million citizens in digital skills and develop 500,000 technology professionals by 2030 [11].

Moreover, the majority of current literature relies predominantly on secondary data, such as literature reviews and performance metrics, rather than employing direct, in-person interviews or field assessments. Prior research in software engineering evaluation indicates that primary data collection methods, including in-person interviews, can capture richer, context-specific insights into technical skills and team dynamics than secondary data alone [12]. In addition to this, understanding the specific cultural characteristics of EA and how these characteristics align with or diverge from European norms, is important to developing effective outsourcing strategies that aid in good collaboration.

The original problem addressed in this thesis is therefore twofold. First the neglect of African contexts in GSD research has led to a large knowledge gap regarding the practical experiences and capabilities of these software engineers, particularly in EA. Second, the reliance on secondary data limits our ability to capture the more nuanced and context-specific realities of technical skills, communication, and process efficiencies in these emerging markets. This study defines the problem as the need to empirically evaluate and compare the performance, cultural dynamics, and organizational practices of software engineers in EA with those in more established European contexts.

1.3 Method used

The research uses a pragmatic mixed-methods approach where both qualitative and quantitative data is used. This method was chosen to give a balance between both detailed insights as well as measurable results. The data collected consists of interviews between 60-120 minutes long, with developers from Sweden, Netherlands, Uganda and Rwanda. The number of developers interviewed range between 10 to 15 from each country. The study incorporates specific technical tasks alongside more open-ended discussion about perceptions and experiences, which matches the semi-structured approach where the interviewer ”has the autonomy to explore pertinent ideas that may come up in the course of the interview” [13]. The study also focuses on methodological flexibility, which has been deemed important by seminal work when researching social issues [14–16].

The interviews collect developer perceptions of their own abilities and other countries present in the study regarding software engineering, problem-solving, and communication styles. Furthermore, the participants are asked to give time estimates for different types of tasks in the software development lifecycle.

The developers go through three technical assessments.

The first assessment is a programming assignment, which is a variation of the two-sum problem. This measures their algorithmic thinking, coding efficiency and practical software development skills. The analysis is made using Code Bidirectional Encoder Representations from Transformers (codeBERT) for embeddings, Principal Component Analysis (PCA) for dimensionality reduction and K-means clustering to identify coding pattern similarities, as well as manually inspecting the code and comments said during the interview.

The second technical task was a system design challenge where participants were told to design an URL shortening service. The task is analyzed manually based on architectural design, scalability, data handling and communication skills.

The last technical task was a code review challenge that looked at both communication styles and security knowledge. The task was to identify vulnerabilities and give feedback accordingly. The sentiment of the feedback was analyzed using Valence Aware Dictionary and Sentiment Reasoner Valence Aware Dictionary and Sentiment Reasoner (VADER) from the Natural Language Toolkit (NLTK) package [17].

1.4 Goal and Purpose

This thesis aims to deepen the understanding of GSD by empirically comparing software engineering competencies and collaboration practices between EA and NWE. The findings are intended to inform both industry and academia. The industry can draw on the results to complement outsourcing strategies and manage cross-cultural teams more effectively, while researchers from academia gain fresh empirical data to strengthen theoretical perspectives on GSD, as well as an analysis between practices across these regions. Furthermore, this work also addresses ethical and social considerations by advocating more equitable and inclusive practices in the global tech sector by laying forward data for hiring considerations, which ultimately contributes to a more just and representative digital society.

1.5 Delimitations

This study is defined by several boundaries that shape its scope and inform how the results should be interpreted. First, it focuses exclusively on software engineers in EA (Rwanda and Uganda) and NWE (Sweden and the Netherlands). Other regions and outsourcing destinations fall outside its scope, which may limit the broader relevance of the findings for the regions specifically.

Second, the sample size is constrained to about 10-15 engineers per country, due to time and resource limitations. This relatively small pool may restrict how representative the results are for each region and its software engineering ecosystem. If it were to act as a purely quantitative study, the results would be non-satisfactory [18], but due to the mixed-methods approach taken, the results are deemed reliable. This is further discussed in section 3.2.

Third, the research centers around coding skills, software architecture, security knowledge, code comment sentiments, communication effectiveness, and organizational culture. It does not address other factors important to a software engineer such as business acumen, how well they innovate, version control, and continuous learning ability. This was due to it being beyond the scope of a masters thesis, and has already been researched over long periods of time for other parts of the world [19].

Additionally, the data relies on insights that are self reported and in-person interviews. While these methods give good insights, they can also introduce biases such as social desirability or recall bias [20, 21]. The combination of quantitative and qualitative data given from the interviews aims to paint a comprehensive picture of current practices. However, the study is not longitudinal, but is instead cross-sectional and therefore it gives a “snapshot” of the current state of software engineering in these countries, rather than tracking changes over time. This study also aims to complement data heavy studies, as empirical ones like this are rare due to the significant time and resources they require.

Finally, although the project explores cross-cultural collaboration, it does not include controlled experimental setups to establish causality between cultural factors and software development outcomes.

1.6 Structure of the thesis

Chapter 1 presents the research problem, objectives, research questions, methodology overview, results summary, and delimitations of the study. Chapter 2 covers the theoretical foundations of GSD, cross-cultural collaboration, emerging tech hubs in EA, and analytical techniques used in software engineering research. Chapter 3 covers the research paradigm, data collection approaches, assessment techniques, and analytical methods used to compare software engineers across regions. Chapter 4 explains the findings from the technical assessments, code review analysis, security knowledge evaluation, and perception studies that address the research questions. Chapter 5 goes into the implications of the findings, focusing on technical skills disparities, communication differences, process optimization strategies, and perception biases. Chapter 6 summarizes insights, discusses limitations, and proposes directions for future research in cross-cultural software engineering studies.

Chapter 2

Background

This chapter provides background information and a review of related work to establish the context for this study. It goes into the theoretical foundations of GSD, which here consists of the trends in the field, as well as the challenges that arise due to geographical, temporal, and cultural differences. The chapter also discusses the need for comparative research in GSD and highlights the underrepresentation of EA and African tech hubs in current research.

The first major area examined is the trends in GSD, where a major trend is going from co-located teams to distributed ones, with multicultural collaboration across many countries. It highlights the economic drivers behind outsourcing and ways that facilitate global collaboration. Additionally, the section addresses the how complex cross-cultural collaboration is due to the impact of communication styles, organizational norms, and unconscious biases present in teams.

The chapter then provides an in depth analysis of emerging tech hubs in EA, particularly Rwanda and Uganda, which are experiencing rapid growth of their technology ecosystems due to large investments in IT infrastructure. It also identifies a critical gap in empirical research on software engineering practices in these regions, justifying the comparative approach of this study.

The second major area explored are analytical techniques and methodologies used in software engineering research. It covers quantitative methods such as standardized coding tests and technical interviews, as well as qualitative methods such as in depth interviews and field assessments. The section also reviewed advanced data analysis techniques, including Machine Learning (ML), Natural Language Processing (NLP), and process visualization, as they are techniques that are increasingly used to evaluate technical skills, communication effectiveness, and workflow efficiencies.

2.1 Global Software Engineering and Cross-Cultural Collaboration

This section introduces the context of GSD, highlighting the evolution of distributed teams and outsourcing. It also goes through the challenges present which are due to geographical, temporal, and cultural differences. The section presses on the need for comparative research in cross-cultural settings and discusses the underrepresentation of the state of software development in EA in current research.

2.1.1 Global Software Development Trends

GSD has has historically been conducted in house and in a co-located space. Nowadays, projects that once were confined to a single location have through advances in ICT allowed organizations to recruit talent from all around the world [22]. Innovations such as high-speed internet, cloud-based collaboration tools, and agile methodologies have contributed to this transition which enables teams to work together in almost real-time despite being separated by geography [23–25].

The economic and strategic drivers behind this shift are also very clear. Rising local labor costs and increasing global competition have given companies incentive to outsource development work to regions that offer advantages in both cost efficiency and access to expertise [26], as shown in Figure 2.1 [26]. However, challenges remain in terms of communication, coordination, and cultural integration even within well-established markets which can lead to various problems such as misunderstandings between team members, delays, and decreased productivity [27, 28].

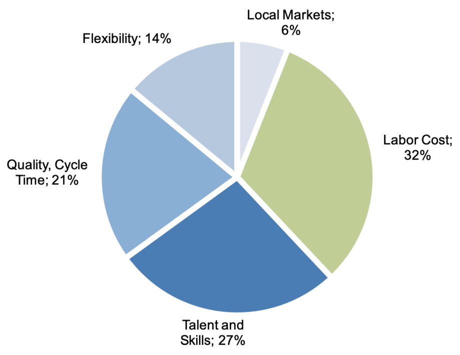

Figure 2.1: Reasons for outsourcing and offshoring.

The literature also highlights the role of agile practices in facilitating global collaboration. Agile methodologies have been adapted to work with distributed environments. However, their implementation struggles to work past the challenges of time-zone differences and cultural diversity. Digital collaboration platforms and automated integration tools as well as scrum methodologies are further allowing for a more efficient global development workflow [29].

The current trends indicate that while distributed development offers benefits such as cost savings and access to specialized skills, it also needs to overcome the challenges of communication and coordination across diverse cultural contexts. These issues are especially highlighted in countries that are the norm of outsourcing, for example China, which presents issues in culture and communication [30]. Research has shown that geographical, temporal, and cultural distances can hinder effective communication and coordination as well as trust, which in turn ultimately impacts project outcomes, highlighted by Table 2.1.

Table 2.1: GSD Challenges Due to Distances [31]

Distance	**Challenges**
Geographical	Lack of informal communication, less shared project awareness, problems in information exchange, knowledge management, process transparency, high communication cost, and coordination issues.
Temporal	Limited synchronous communication and delay in feedback.
Socio-cultural	Inconsistency in work practices, less informal communication, different terminologies, diverse hierarchy, and differences in work ethics.

2.1.2 Cross-Cultural Collaboration in Software Engineering

How effective GSD is is influenced by the extent of how well cross-cultural collaboration works in teams. Cultural variations such as communication styles and organizational norms affect how team members interact with each other and coordinate their work. Power distance, individualism/ collectivism, and uncertainty avoidance show how communication styles influence organizational practices in global teams [32, 33].

Hofstedes Cultural Dimensions is a framework developed to understand how cultural differences across nations affect behavior in organizations and society. Particularly relevant to the software industry, the research began with IBM employees across different countries in the 1970's. The framework introduces concepts such as power distance, uncertainty avoidance and individualism vs collectivism.

Power distance refers to the extent less powerful members of an organization accept and expect that power is distributed unequally. In high power distance cultures it is seen that hierarchical differences are accepted as normal, while in low power distance cultures the members strive for power equalization and also demand justification for power inequalities if present.

Uncertainty avoidance indicates to what extent a culture programs its members to feel comfortable or uncomfortable in unstructured situations. Uncertainty avoiding cultures try to minimize unstructured situations through strict rules, laws, and behavioral codes, while uncertainty accepting cultures are more tolerant of different opinions and behaviors.

The dimensions of individualism vs collectivism measure the degree to which people in a society are integrated into groups. Individualist societies have loose ties between individuals, with everyone expected to look after themselves and their immediate family only. Collectivist societies instead consist of strong in-groups (often extended families) that protect members in exchange for loyalty [33].

It is know that high-context communication styles that normally are seen in collectivist cultures can lead to misunderstandings when they interact with low-context communicators, in turn impacting team coordination and the effectiveness of the collaboration between the teams [34]. Therefore, one important factor that the thesis aims to uncover is the similarity in communication styles between the two regions of NWE and EA.

Studies indicate that cultural differences not only influence communication but that they also affect how effective distributed team are. As misinterpretations arise from cultural differences, they lead to inefficiencies in work as well as lowering trust between individuals [27]. Especially differences in power distance and uncertainty avoidance can create barriers to open communication, in turn reducing efficiency of collaboration and lastly impacting overall team performance [35].

A study showed that African and European stakeholders demonstrated opposing communication styles, as well as that Africans adhere to higher power distance norms, which means that decision making is centralized and follows a strict hierarchy, while Europeans tend to have more flexible power structures, leading to potential misalignment in expectations and collaboration [36].

Research has also looked into how unconscious biases impact team dynamics. In environments characterized by high power distance or that are collectivist, these biases could result in hierarchical communication patterns or misaligned expectations, which in turn hurt collaboration [37]. An engineer from a society with high power distance will in turn struggle to adapt to the dynamics of a non-hierarchical company.

While traditional cross-cultural research has often addressed these issues at a broad level, there is also research pointing out that even minor, unconscious biases can significantly affect technical collaboration [38]. A deeper understanding of these nuances is important for outsourcing strategies that aim to have seamless communication and coordinated work practices among globally distributed teams [39].

2.1.3 Emerging Tech Hubs in East Africa

Recent investments in IT infrastructure have become a characteristic of the EA technology landscape. More specifically, it has been seen that Rwanda has significantly enhanced its broadband connectivity, digital services, and regulatory frameworks for IT, which all signals them driving the growth of their technology ecosystem [40, 41]. National policies aimed at knowledge transfer and startup support have also aided this growth, which is positioning Rwanda as a tech hub in EA [42]. This is also shown in the GDP growth of Rwanda, where the ICT sector contributed to a 35% growth in Rwanda's GDP in 2023 [43]. IT investments in Uganda are also progressing, but at a slower pace compared to Rwanda [44]. This also highlights a natural contrast between the two countries technological landscapes.

Despite these rapid developments, empirical research directly assessing the experiences of software engineers in EA, specifically Rwanda, is notably limited. The gaps in evaluating technical skills, communication practices, and work methodologies among software engineers in these regions have been noted previously [45, 46]. This lack of empirical evidence is particularly striking because of the rapid growth in these countries driven by increasing internet penetration and a surge in local entrepreneurial activity [47]. Though noted by some entrepreneurship hubs such as Norrsken which have setup a co-working space in Kigali [48], this trend has not yet been picked up by researchers.

2.2 Analytical Techniques and Methodologies in Software Engineering Research

This section outlines the research methods and analytical techniques that are relevant to this study. It covers both quantitative and qualitative approaches. It also covers data analysis methods such as ML, NLP, and process visualization.

2.2.1 Quantitative Assessment Methods

Quantitative assessments are used in software engineering research to evaluate technical competencies and problem-solving abilities [49]. Standardized coding tests provide objective metrics, and are also useful for assessing programming proficiency [50], while challenges focusing on bug detection and security, as well as system design offer insight into practical problem-solving skills of developers [51].

2.2.2 Qualitative Methods and Field Assessments

Qualitative methods uncover the context behind the numbers from the quantitative assessments [52]. They reveal the contextual and experiential factors that shape software engineering [53]. In-depth and in-place interviews as well as field assessments highlight the nuances of real-world environments [52], from hierarchical communication to management approaches.

Research underscores the value of combining qualitative and quantitative data for a holistic understanding of empirical research in a mixed-methods approach [54].

2.2.3 Pragmatic Mixed-Methods Research Approach

A pragmatic mixed-methods approach puts together quantitative assessments, such as standardized coding tests and perception studies, with qualitative methods such as interviews and sentiment analysis of communication patterns.

A pragmatic mixed-methods approach does not follow a theoretical approach, but keeps to a grounded reality of practicality and contextual responsiveness [55]. This method encompasses this paper, as it is based on empirical data taken directly from the source.

2.2.4 Evaluating Software Engineering Skills

The evaluation of the technical competencies of software engineers can be seen as an important area of research in software engineering due to interest from both hiring managers and interviewees. Various methods have been developed to assess and rank developer skills, which will be discussed in the related works section. These range from standardized coding tests and technical interviews to more advanced approaches utilizing ML and data analytics. These methods aim to provide objective metrics for comparing programming proficiency, problem-solving abilities, and system design skills across different regions and organizational contexts.

Technical interviews can complement coding tests, and research indicates that structured technical interviews are effective in evaluating both technical and soft skills, including teamwork and communication [56]. This approach allows for a more holistic evaluation of a developer competencies and in turn enables more accurate benchmarking across different regions.

Machine Learning and Data Analytics in Skill Assessment

Advancements in data analysis techniques have provided a complement to traditional methods of analysis in software engineering. (ML) and NLP are increasingly applied to analyze code comments, communication logs, and other textual artifacts [57]. Transformer-based models such as Bidirectional Encoder Representations from Transformers (BERT), have been used to generate code embeddings that enable clustering and comparison of coding styles and communication pattern using k-means clustering [57–59].

Additionally, clustering analyses have been used to compare programming strategies and behaviors across varying skill levels [60], and therefore being relevant in showing how cultural contexts influence problem-solving strategies and communication styles.

codeBERT is a pre-trained model designed specifically for programming languages and natural language [61]. It adapts the BERT architecture to code-related tasks which allows it to learn meaningful representations from both code and text. The model builds on research about how language models capture structural information. BERT models form ”a rich hierarchy of linguistic information” [62], and codeBERT extends this capability to programming languages.

PCA is a dimensionality reduction technique used to transform high-dimensional code embeddings into lower-dimensional representations while preserving maximum variance in the data. In code similarity analysis PCA plays an important role in visualizing relationships between code fragments that would otherwise remain hidden in high-dimensional embedding spaces. PCA is described as particularly valuable for ”initial exploration” of code embeddings as it helps to identify broad patterns in how different code implementations relate to each other [63]. This makes it an essential first step in analyzing the relationships between code fragments represented as high-dimensional vectors, since code embeddings typically contain hundreds of dimensions (codeBERT uses 768-dimensional vectors [62]), making direct visualization impossible.

Sentiment analysis is a branch of NLP, and it plays an important role in understanding the emotional tone behind textual communication, including within software engineering environments. One widely used NLP tool for sentiment analysis is VADER [17]. It is integrated into the (NLTK) library, and VADER was from the beginning designed for analyzing sentiments in social media and other forms of informal communication [17]. It combines a sentiment lexicon of over 7,500 features with rule-based heuristics that takes linguistic features such as punctuation and capitalization into consideration, and requires no training data [17]. VADER uses a normalized compound score that ranges from most negative (-1) to most positive (+1), which allows for a straightforward assessment of overall sentiment [17]. Even though VADER is not specifically designed for software engineering purposes, it has been used in this context previously [64].

2.3 Related Work

This section reviews the existing literature that is most relevant to this study. It covers prior research on GSD, cross-cultural collaboration, and the evaluation and analysis of software engineering skills. It identifies both the contributions and the limitations of existing studies, and thereby justifying the research focus of this project.

2.3.1 Related Work on Global Software Development

GSD is characterized by distributed teams collaborating across geographical and cultural boundaries. Prior research has examined the challenges associated with communication, coordination, and cultural integration in GSD contexts [31]. For instance, challenges in multilingual communication and the impact of cultural diversity on collaboration have been explored in studies focusing on global software teams [65].

Research also highlights the importance of understanding cross-cultural dynamics to enhance collaboration and productivity in distributed teams [66]. The studies emphasize the role of multilingualism and cultural competencies in mitigating communication barriers and improving team interactions [67]. However, most work on outsourcing and GSD reviewed focuses on major technopoles and emerging centers such as India, Ireland, Israel, Russia, The Philippines, China, Eastern Europe, North Africa, Brazil and Chile in South America [4, 6, 68, 69], leaving a noticeable gap in empirical research on emerging markets such as Rwanda and Uganda in EA. Furthermore, a literature review on information technology outsourcing looked at 91 different papers on outsourcing published in 51 unique journals, and found no research published by any country in Africa specifically [70], indicating a lack of engagement from African parties in this regard as well.

The studies reviewed that cover Africa, focus on more developed countries, where the main focus is often South Africa, Kenya, Nigeria, North African countries and even Mauritius in one study [7–10]

This study addresses that gap by conducting a comparative analysis of software engineers in Rwanda, Uganda, Sweden, and the Netherlands, offering new insights into how cultural and organizational contexts shape software engineering practices, as well as benchmarking the skills of these engineers, potentially putting Rwanda and/or Uganda on the map for discussions related to outsourcing in Africa.

2.3.2 Related Work on Analytical Techniques

Analytical techniques, including ML, NLP, and process visualization, are applied in software engineering research. These methods enable the extraction of insights from large datasets, such as code repositories and communication logs. For example, embedding models and clustering techniques have been used to evaluate coding styles and problem-solving approaches across cultural contexts [71].

NLP techniques, such as sentiment analysis and topic modeling, are employed to analyze communication styles and collaborative behaviors in software teams [72]. There are also studies focusing on how to leverage process visualization techniques specifically for different software engineering processes [73, 74]

2.3.3 Related Work on Assessing Software Engineers

Standardized coding tests are widely used for evaluating fundamental programming skills, such as algorithmic thinking, data structures, and problem solving [50], especially in the industry [75–77]. Studies have also looked at general skills needed for software engineers [56], as well as a larger competency framework developed by SWECOM [78], which has a vast number of references to related articles. The assessments also include incorporating bug detection challenges and system design tasks that provide more holistic assessments of practical problem-solving and architectural skills [78, 79]. Lastly, this study takes inspiration from seminal work on methodological flexibility when to comes to researching social issues [14–16].

2.3.4 Explicit Research Gaps and Summary of Contributions

Although existing research provides valuable insights into GSD and cross-cultural collaboration, it predominantly focuses on established tech hubs in Asia, South America, and Europe. There is a noticeable gap in empirical research on emerging markets, particularly in EA. Especially, there is no empirical research regarding GSD or skills assessments of engineers in Rwanda and Uganda. Even so, going broader, empirical research on developers and engineers generally in Rwanda and Uganda is comparably low.

This study addresses these gaps by conducting a comparative analysis of software engineers in Rwanda, Uganda with those in Sweden and the Netherlands. It uses methods such as NLP, sentiment analysis, and process visualization to do the analysis. Also, it contributes to the academic discourse by providing empirical data from underrepresented regions, offering new insights into cross-cultural collaboration. Using a pragmatic mix-methods approach, it closes the gap between productivity metrics and cultural nuances.

By addressing these research gaps, this study provides a more comprehensive understanding of GSD practices, informing strategic decisions in global collaboration and outsourcing.

Chapter 3

Methods

This chapter outlines the methods used to compare software engineering practices between engineers from EA (Rwanda and Uganda) and NWE (Sweden and the Netherlands). It describes the overall approach taken in the study, including data collection, experimental design, analytical methods, and how the reliability and validity of the findings were ensured.

3.1 Research Paradigm

This study uses a pragmatic mixed-methods approach which combines both quantitative and qualitative methods for evaluating software engineering skills clearly and comprehensively [80], and in turn understanding cross-cultural differences between the regions in the study. The framework was selected because it connects the objectivity and measurability of quantitative data with the insights and understanding you get from qualitative studies [80]. This approach enables thorough analysis of technical skills, communication patterns, and organizational practices across different cultures.

Other methodological options were also considered. A purely quantitative method was not selected because it typically focuses only on comparisons or relationships between variables and could in turn miss important cultural nuances and contextual details. On the other hand, a purely qualitative method could be valuable for uncovering processes, experiences, and contextual depth, but this could also not be suitable since it may lack the ability to generalize results or make clear comparisons of technical performance across groups. Therefore, a mixed-method approach was identified as the most appropriate strategy because combining both quantitative and qualitative approaches can offset the inherent limitations of each method individually, which in turn allows for more balanced and insightful analysis which still maintains both clarity and depth [80]. Pragmatism also allows for greater depth and breadth in the research [81] and frees researchers from the ”tyranny of method” [81, 82]. Lastly, seminal work has argued that science progresses best when not constrained by rigid methodological rules [14], especially when addressing the complexity of social problems [15], which is a theme followed in the methodology.

Research Process

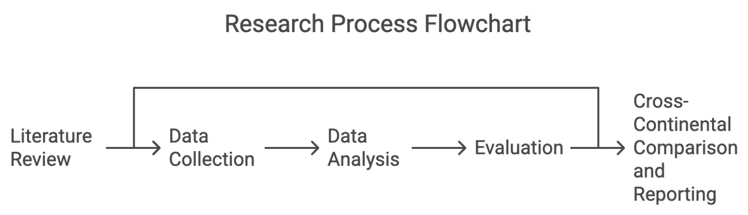
Research Process Flowchart
Literature Review
Data Collection
Data Analysis
Evaluation
Cross-Continental Comparison and Reporting

Figure 3.1: Research Process Flowchart

Figure 3.1 illustrates the iterative nature of the research process, emphasizing feedback loops between data collection, analysis, and evaluation.

3.2 Research Questions and Design

The research questions were formulated based on gaps identified in the literature regarding GSD and cross-cultural collaboration with particular attention to the limited representation of EA contexts [83]. Three main research questions guide this study.

RQ1: How do software engineers in East Africa and North Western Europe differ in coding skills and problem-solving strategies?

RQ1 is addressed through quantitative methods, including standardized coding and system design tests designed to measure programming skills and complexity of solutions, where analysis consists of clustering techniques for comparative analysis. Additionally, qualitative methods involve in-depth interviews to explore the reasoning behind the strategies.

RQ2: How do communication styles and organizational practices differ between regions?

RQ2 is answered through sentiment analysis on code comments and way of expression when solving problems. Quantitative data will show differences in process steps across the countries. Furthermore, qualitative data such as interviews with team-leads are also analyzed to compare organizational cultures.

RQ3: What biases exist across the countries and how would this affect collaboration between software engineers from East Africa and North Western Europe?

RQ3 employs perception study to quantitatively assess biases, as well as conversing with the engineers about their answers to the questions. Furthermore, interviewees as later stages will be asked to give their reflections on the results of the perception study.

Data Collection, Validity & Reliability

3.2.1 Data Collection

Participants were selected to ensure representation from diverse levels of seniority. The sample includes 48 software engineers, with 10–15 participants each from Rwanda, Uganda, Sweden, and the Netherlands. Interviews typically lasted 60–120 minutes. The participants were allowed to use a Integrated Development Environment of their own choice, but were not allowed to access the internet or use AI tools.

3.2.2 Data Validity

These numbers (48 software engineers, with 10–15 participants from each country) are adequate because saturation in qualitative research, defined as ”the point in data collection when no additional issues or insights are identified and data begin to repeat so that further data collection is redundant” [84], typically occurs within 9–17 interviews for relatively homogeneous populations [84]. Software Engineers fall under the category of being a homogeneous population [85], and thus having saturation reached at the number of participants included in this study per country. As a whole, software engineers from 4 different countries is a heterogeneous group, and 48 in depth interviews is deemed more than sufficient, especially since research also points to that saturation is reached within a similar number of interviews for a heterogeneous population as a homogeneous one [84]. Furthermore, a study using semi-structured, open-ended interview questions in Ghana noted that after 30 interviews, their codebook contained 109 content-driven codes. Importantly, 80 (73%) of these codes were identified within just the first six interviews, and an additional 20 codes were found in the next six interviews - meaning 92% of all codes were identified within the first 12 interviews. The authors concluded that ”Based on our analysis, we posit that data saturation had for the most part occurred by the time we had analyzed twelve interviews” [86].

Furthermore, the sampling was made through professional networks, and the sample members were not selected at random from any population, and instead were selected because they were the easiest to recruit for the study. This is referred to as convenience sampling, andis often used to recruit participants to a study as well as used in conjunction with most study designs [87].

For the quantitative part of this study, the sample size provides limited statistical power, especially at the country level, and especially if only considered without the qualitative component. Using standard Confidence Interval (CI) calculations at a 90% confidence level, a margin of error can be determined for proportions using the formula [88]:

$$\text{Margin of Error} = z \times \sqrt{\frac{p(1-p)}{n}} \tag{3.1}$$

Where z is the z-score (1.645 for 90% confidence level), p is the estimated proportion (0.5 provides maximum variance), and n is the sample size [89]. Applying this formula to our samples:

For individual countries:

$$\text{Rwanda (n=13):} = 1.645 \times \sqrt{\frac{0.25}{13}} = \pm 22.8\%$$

$$\text{Uganda (n=14):} = 1.645 \times \sqrt{\frac{0.25}{14}} = \pm 22.0\%$$

$$\text{Sweden (n=10):} = 1.645 \times \sqrt{\frac{0.25}{10}} = \pm 26.0\%$$

$$\text{Netherlands (n=11):} = 1.645 \times \sqrt{\frac{0.25}{11}} = \pm 24.8\%$$

For regional comparisons:

$$\text{East Africa (n=27):} = 1.645 \times \sqrt{\frac{0.25}{27}} = \pm 15.8\%$$

$$\text{North Western Europe (n=21):} = 1.645 \times \sqrt{\frac{0.25}{21}} = \pm 17.9\%$$

For the full sample:

$$\text{Total (n=48):} = 1.645 \times \sqrt{\frac{0.25}{48}} = \pm 11.9\%$$

These wide CI at the country level (over ±20%) show that there is uncertainty in the quantitative findings for individual countries. This limitation is both acknowledged and addressed through the mixed-methods approach taken by the study where quantitative data is primarily used for identifying broad patterns and insights from qualitative data provide the depth and context for country specific analysis. This statistical limitation is consistent with the exploratory nature of this cross-cultural comparative study, where depth of understanding often takes the driver seat over statistical precision [90]. Also, mixed-methods research can compensate when the quantitative approach by itself is inadequate to develop a complete understanding about a research problem or question [91].

However, while wider CI indicate less precision in quantitative estimates, they still provide valuable information about the range of plausible values and uncertainty in the data. As emphasized by Hespanhol et al. [92] decision making should not be based only on ”the dichotomized interpretation of confidence intervals (i.e., statistically significant or non-statistically significant),” but rather should incorporate ”a more in-depth analysis and interpretation of the values and width (i.e., precision) of CI”. This principle aligns with mixed-methods approach taken that is looking to balance statistics with contextual understanding. It is therefore important to recognize that quantitative measures with wider confidence intervals can still give valuable insights when complemented by qualitative data that provides depth and context. Furthermore, wider confidence intervals are an expected consequence of the design choices made in pragmatic research that prioritize real-world applicability [93], once again aligning with the approach taken in the study.

To further ensure the validity of the data, all assessments and interview questions were designed to reflect real world software engineering tasks and communication practices. The type of questions was chosen based on the skills that the scientific literature deems important for software engineers and were refined by consulting experienced professionals within the field. Since the literature is split on this issue, and in the pragmatic essence of this study, this paper chooses to go with a study that also references the ten knowledge areas of the Software Engineering Body of Knowledge (SWEBOK). The areas chosen from that list were Software Design, Construction, Testing, Quality, Process, Maintenance, which covers 6 of the 10 skills listed for software engineers [94].

Furthermore, a study showed that what classifies a good developer is their technical and social skills [95]. In technical skills, they look at coding ability, which asks how proficient is the knowledge of an individual, and how good their ability to code is, and the second aspect is the quality of work, which asks how good is the code that an individual produces? Social Skills are soft skills that measure the ability to work as an individual and in teams. Three important skills are collaboration proficiency, project management ability, and motivation [95]. These were not taken into account due to the scope of the study.

3.2.3 Data Reliability

Reliability refers to the consistency of the results, and in this study, it was maintained by administering the same standardized assessments to all participants. Every interviewee was given the same tasks under the same conditions as well as the same amount of time. The coding challenges and system design tasks are scored using predefined criteria to keep consistency across all participants. Pilot tests were conducted before the main study to make sure that assessments were clear and that the questions were interpreted correctly by all participants. However, due to geographical reasons, the pilot test was done on Swedish participants only.

To ensure that the data is accurate and truthful, the data is collected directly from the participants during the coding challenges, system design tasks, and interviews. This direct approach minimizes the risk of misrepresentation or manipulation of results. During the case of a virtual interview, a second person was to be present to validate the integrity of the candidate.

The coding tasks are monitored to ensure that the solutions are created independently by the participants, and notes are taken during the assessments to capture the whole thought process of the candidate and any deviations from the assessment rules.

3.3 Assessments and Challenges

Programming Assessment: Unique Pairs that Sum to Target

The programming assessment chosen for this study is the problem titled **Unique Pairs that Sum to Target**. The problem is defined as follows:

> Given a list of integers `nums` and an integer `target`, return a list of unique pairs (a, b) where:
>
> - $a + b =$ `target`
> - (a, b) is considered the same as (b, a), so no duplicate pairs should appear.
>
> You may return the pairs in any order.

Example 1:

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

Example 2:

```
Input: nums = [2, 2, 4, 4], target = 8
Output: [(4, 4)]
```

Multiple approaches can solve this problem, each with different time and space complexities. These approaches are detailed below:

Brute Force Approach

- **Description:** The brute force method involves checking all possible pairs in the list to see if their sum equals the target value.
- **Algorithm:**
  1. Iterate over each element i in `nums`.
  2. For each i, iterate over every subsequent element j.
  3. If $nums[i]+nums[j] =$ `target`, add the pair $(nums[i], nums[j])$ to the result set.
  4. Use a set to store pairs to automatically handle duplicate pairs.

- **Time Complexity:** $O(n^2)$ since all pairs are checked.
- **Space Complexity:** $O(n)$ for storing the result set.
- **Example Implementation (Python):**

Listing 3.1: Brute Force Approach for Two Sum Problem

```python
def find_pairs(nums, target):
    pairs = set()
    for i in range(len(nums)):
        for j in range(i + 1, len(nums)):
            if nums[i] + nums[j] == target:
                pairs.add((min(nums[i], nums[j]),
                           max(nums[i], nums[j])))
    return list(pairs)
```

Sorting and Two-Pointer Technique

- **Description:** This approach sorts the list first, then uses a two-pointer technique to find pairs.
- **Algorithm:**
  1. Sort the list.
  2. Initialize two pointers: one at the start (left) and one at the end (right).
  3. If the sum of the two pointers equals the target, add the pair to the result and move both pointers.
  4. If the sum is less than the target, move the left pointer forward.
  5. If the sum is greater than the target, move the right pointer backward.
- **Time Complexity:** $O(n \log n)$ for sorting, and $O(n)$ for the two-pointer search, resulting in an overall complexity of $O(n \log n)$.
- **Space Complexity:** $O(n)$ for storing the result set.
- **Example Implementation (Python):**

Listing 3.2: Two-Pointer Technique for Two Sum Problem

```
def find_pairs(nums, target):
    nums.sort()
    left, right = 0, len(nums) - 1
    pairs = set()

    while left < right:
        current_sum = nums[left] + nums[right]
        if current_sum == target:
            pairs.add((nums[left], nums[right]))
            left += 1
            right -= 1
        elif current_sum < target:
            left += 1
        else:
            right -= 1
    return list(pairs)
```

Hash Set Method (Optimal Solution)

- **Description:** This approach uses a hash set to check for the complement of each element in a single pass.
- **Algorithm:**
  1. Initialize an empty set to store seen numbers.
  2. Iterate over the list. For each number, calculate its complement as `complement` = `target` − `num`.
  3. If the complement is in the set, add the pair to the result set.
  4. Add the current number to the set.
- **Time Complexity:** $O(n)$ due to a single pass through the list.
- **Space Complexity:** $O(n)$ for storing the result set and the hash set.
- **Example Implementation (Python):**

Listing 3.3: Hash Set Method (Optimal Solution) for Two Sum Problem

```
def find_pairs(nums, target):
    seen = set()
    pairs = set()

    for num in nums:
        complement = target - num
        if complement in seen:
            pairs.add((min(num, complement),
                       max(num, complement)))
        seen.add(num)

    return list(pairs)
```

This problem is a remade version of the two-sum problem, which is the most practiced problem on Leetcode [96]. This particular problem was selected due to its suitability in assessing core software engineering competencies such as algorithmic thinking and coding efficiency, and understanding of data structures. The task is complex enough to effectively differentiate skill levels among participants yet straightforward enough to be completed within a reasonable time frame. Additionally, this type of algorithmic challenge is commonly employed in software engineering recruitment contexts which enhances the ecological validity and practical relevance of the assessment [97].

Analysis for Unique Pairs

To analyze code patterns across the different regions ML techniques for clustering code similarities together with manual inspection of algorithmic approaches was used.

The codeBERT model specifically designed for programming language understanding was used. Each code submission was processed by first creating an embedding. Here code snippets were tokenized and passed through codeBERT to generate high-dimensional vector representations (embeddings) that capture semantic and structural properties of each submission. Then dimensionality reduction was used. This resulted in high-dimensional embeddings (768 dimensions), that were reduced to two dimensions using PCA to easier visualize the results and in turn conduct the analysis. Lastly, K-means clustering was used, where k=4 was applied to the embeddings to find patterns in the code and see which countries use what approaches. The algorithm partitioned the code submissions into clusters based on similarity in the embedding space. This was used to generate Figure 4.1 in the results, by using PCA 1 and PCA 2 for the two dimensional axis.

The distribution of clusters across countries were also represented using stacked bar charts, allowing for direct comparison of coding patterns between regions.

In addition to automated analysis the results were also manually inspected to classify whether the engineers used a hash map solution, two-pointer solution, nested loops or failed the task. Notes were also taken during the process to see if the engineers mentioned another more effective solution, as well as encouraging them in their explanations to continue speaking.

System Design Challenge - URL Shortening Service

This study includes a system design challenge which is to design a URL Shortening Service, similar to commonly used services like Bit.ly. The challenge requires participants to design a system that can:

- Handle a large number of requests efficiently.
- Store and retrieve shortened URLs effectively.
- Ensure each shortened URL is unique.
- Redirect users accurately from shortened URLs to the original destinations.

This system design challenge was used to assess architectural design, scalability and data handling. Participants completed the task in spoken language, explaining their design choices and reasoning in real time. This allowed for evaluation of both technical understanding and communication skills. The challenge reflects practical, real-world scenarios and provides a fair basis for comparing engineers across regions, and in this case EA and NWE. Studies show that software engineers often lack practical system design skills, particularly in scalability and architectural thinking [98]. This challenge addresses this skills gap by looking at if they can design a scalable and maintainable system.

Analysis for System Design

The results were aggregated using manual techniques, going over each answer and noting down overall country generalizations, based on the four objectives noted in 3.3

Code Review Challenge - Authentication System

The Code Review Challenge is designed to evaluate the participants ability to find security vulnerabilities and to see how they provide feedback. The task involves reviewing and suggesting improvements to a poorly designed authentication system through code comments. The code is presented in Python, but participants are allowed to use Java if preferred.

Other than technical evaluation, this challenge also supports comparative analysis of communication styles and feedback methods among software engineers from EA and NWE. By doing sentiment analysis on the comments, the study hopes to give an unique perspective on how cultural context may influence communication in software engineering.

Assessment Code

The following is the Python code provided to the participants for review:

Listing 3.4: Authentication System Code

```
# This authentication system is designed to be
    simple and efficient
# It provides basic login and password management
    functionality

class AuthSystem:
    def __init__(self):
        self.users = {"admin": "password123"}  #
            This makes it easy to manage users!

    def login(self, username, password):
        # Quick and simple login check
        if username in self.users and
            self.users[username] == password:
            print("Login successful!")  # Users
                will appreciate this feedback!
            return True
        return False  # Should we provide more
            details on failure?

    def change_password(self, username,
        new_password):
```

```
        self.users[username] = new_password  #
            Password updates are quick and direct!

# Let's test the system!
auth = AuthSystem()

auth.login("admin", "password123")  # Works well!
auth.change_password("admin", "newpass")  # Very
    easy to change passwords!
auth.login("admin", "newpass")  # Success message
    confirms everything is good!
```

After the issues are identified, the participants are encouraged to leave a code review, as well as change any comments they deem unnecessary, and add comments the way they feel are necessary. Code reviews are recognized as a way for reducing software defects and improving the quality of software projects [99]. The assignment is evaluated based on the tone of comments and the review, and if the reviewer noticed any security vulnerabilities. The reason for including an aspect of security is because even though software engineers are not specifically experts in security, they are held responsible for developing secure applications [100].

Security Issues Analysis

There were 11 distinct security issues that engineers identified, which were cataloged:

- Plain text password storage
- Hardcoded credentials
- No authentication for password change
- Information leakage in error responses
- Inappropriate logging/print statements
- No input validation for passwords
- Unsafe password comparison
- No rate limiting for login attempts

- Password complexity requirements
- Logging of security events
- Two-Factor Authentication (2FA) recommendation

For each identified issue the percentage of engineers from each region and experience level who identified the problem were noted. Engineers were categorized into groups of juniors and seniors from each country, where seniors had over 3 years of experience and juniors under 3 years of experience. Visualization techniques included bar charts, heat maps, and tabular visualizations. Lastly, juniors and seniors were compared explicitly for deeper understanding of the data.

Code Comments and Sentiment Analysis

The varying tones in the comments provide an opportunity for sentiment analysis. Sentiment analysis will be applied to understand the emotional context of the comments. This aims to reveal how engineers express feedback and suggestions. It also contributes to the goal of the study which is comparing communication styles and feedback mechanisms across different cultural contexts.

The code contains various types of comments, which can be categorized as follows:

Comment Category	**Details**
Positive Comments	**Example:** `# Works well!`, `# Success message confirms everything is good!` These comments express positive feedback but lack technical depth.
Neutral Comments	**Example:** `# Quick and simple login check` These comments describe functionality without any emotional tone.
Negative Comments	**Example:** `# WHY would you add this!?` These comments express negative feedback.

Analysis of Code Comments

To analyze cultural differences in code reviewing practices, a structured approach was implemented combining NLP techniques with manual analysis of developer feedback.

For systematic analysis of comments, the process began with comment extraction using regular expressions, where all inline comments were extracted. Comments were then aggregated by country to enable a comparative analysis of commenting patterns across regions. To understand the emotional tone of review comments, sentiment analysis using the VADER sentiment analysis tool from the NLTK library was used. Rather than simply aggregating all comments by country before analysis, the sentiment scores were first calculated for each individual engineers comments. This was as previously all comments were aggregated, but specific engineers swayed the results which turned them incomprehensible and biased. For each engineers comments, four key sentiment metrics were computed, which were Negative, Neutral, Positive and compound scores. Negative sentiment score as 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 [17]. Then the average sentiment was calculated for each country. Overall sentiment was classified for each country based on normal classification thresholds [17]:

- **Positive**: compound score > 0.05
- **Negative**: compound score < -0.05
- **Neutral**: compound score between -0.05 and 0.05

Developer Quantitative Study

To answer RQ3, the interviewees were asked to rate the software development, communication, and problem-solving skills of engineers from all the countries including their own on a scale from 1 to 10, where 1 is the lowest rating and 10 the highest rating. Their comments on this matter were also recorded.

To understand how the development process itself is different between regions and countries, the interviewees were asked to provide a self-assessment of how long they believe various development tasks typically take, as well as giving justifications for why. As it is out of the scope of this study to analyze detailed activity logs, this section is intended to serve as a complementary source of insight alongside other studies based on log data.

The software engineers were told to give a time estimate on the following categories of tasks, which were picked pragmatically after looking at research in the area [101, 102]:

- **Issue Lifecycle**
  - Investigation and root cause analysis
  - Implementation planning and design
  - Development and testing
  - Documentation and pull request
  - Issue verification and closure
- **Code Review Process**
  - Code cleanup and documentation
  - Running tests locally
  - Peer review process
  - Addressing feedback
  - Final approval
- **Deployment Process**
  - Build and test automation
  - Staging deployment and testing
  - Stakeholder review
  - Production deployment
  - Post-deployment verification
- **Bug Fix Process**
  - Bug investigation and reproduction
  - Fix implementation and testing
  - Documentation and review
  - Deployment and verification

Development Process Visualization

To analyze and compare development processes across different regions the study used time estimates provided by software engineers from Rwanda, Uganda, Sweden, and the Netherlands. The methodology focused on three analyses, which consist of process flow visualization, identification of top and bottom steps by time consumption and a critical path analysis.

To ensure consistent analysis across participants who provided estimates in different formats (minutes, hours, days), a time normalization function that converted all time estimates to minutes was implemented to have a common unit for comparisons. This also allowed for analysis across different countries and experience levels. The normalization approach handled various time formats through pattern matching. Weeks were converted to minutes (1 week = 5 days = 40 hours = 2400 minutes), days were converted to minutes (1 day = 8 hours = 480 minutes), and hours and minutes were converted directly. This standardization was needed as participants used varying time formats in their responses.

To visualize the development process flow for each country, directed graphs that represented the sequence of processes and the relative time spent on each were created. The visualization included process nodes where each development process was represented as a node in the graph. The size of the node was proportional to the total time spent on that process per country, making it visually apparent which process consumed the most resources. Nodes were also color coded using a gradient from green to red based on time intensity. The color green represented shorter times and red longer times, to give an immediate visual indication of where time was spent. Arrows between nodes showed the typical sequence of processes in software engineering, going from issue investigation through to deployment.

To identify which development steps consumed the most and least time for each country, a comparative analysis was created where all steps across processes were aggregated for a given country, then average time spent on each step was calculated and sorted by time consumption (highest to lowest). At last a visualization of top 5 and bottom 5 steps were visualized based on average time. This analysis would show which steps in the development process consumed the most and least time per country with an aim to show where engineers from different regions focus their efforts.

To further show potential bottlenecks in the processes, a critical path analysis was made where steps that consume over 10% of the the total development time were found.

Lastly, a cross country comparison was made to show how long each step took for each country, visualizing the data of each country beside each other in a table.

This process visualization approach shows the development workflows [103]. With this the study aims to show differences in work patterns, and strengths and weaknesses within the software engineering lifecycle. It contributes to the goal of the study which is to compare software engineering practices across EA and NWE by providing a analysis of how teams operate. By going into detail in the processes, the study uncovers cultural and organizational influences on development practices.

Chapter 4

Results and Analysis

This chapter presents the findings from the comparative study of software engineers in EA (Rwanda and Uganda) and NWE (Sweden and the Netherlands). The results are organized to address our three research questions:

RQ1 (coding skills differences) is addressed through the programming assessment and system design challenge results.

RQ2 (communication and problem solving variations) is examined through sentiment analysis of code reviews and process time estimations.

RQ3 (biases affecting collaboration) is explored through perception ratings and their implications for cross cultural teams.

Throughout the chapter, the study integrates quantitative measurements with qualitative insights from interviews to provide a comprehensive picture of regional differences in software engineering practices.

4.1 Technical Assessment Results

4.1.1 Unique pairs

The approaches taken to solve the programming assessment in Section 3.3 are distributed as shown in Table 4.1 between the countries.

Rwandan participants mostly relied on using nested loops, but also had a minority applying hash-based optimizations. No submissions from Rwanda used the two-pointer or counter technique, though a few noted that this way to solve the problem is more efficient during the interview process.

In contrast, Swedish software engineers showed strong algorithmic understanding where a majority was seen going for a hash map solution and a small minority using nested loops or two-pointer approaches. Similarly to Swedish engineers, engineers from the Netherlands demonstrated an above average grasp of efficient solutions, with more than half using hash maps/sets and some using two-pointer methods.

Ugandan engineers had a high proportion of submissions classified as failed. Only a small number used nested loops or hashing strategies. The reasoning for this was because many Ugandan engineers opted to cheat by using AI during this part of the test and therefore had their results nullified.

Country	**Hash map/Set**	**Nested For Loops**	**Two-Pointer/Counter**	**Failed/ Other**
Rwanda	3 (23%)	9 (69%)	0 (0%)	1 (8%)
Uganda	4 (29%)	2 (14%)	0 (0%)	8 (57%)
Sweden	8 (80%)	1 (10%)	1 (10%)	0 (0%)
Netherlands	6 (55%)	3 (27%)	2 (18%)	0 (0%)

Table 4.1: Distribution of solution approaches by country

The clustering of codeBERT embeddings in Figure 4.1 reveal groupings of code submissions. Figure 4.2 easier shows how Swedish and Dutch software engineers are very similar in coding styles, but they also have a surprising similarity with Rwandan engineers, while Ugandan engineers differ from the rest, having no submissions in cluster 1 or 3, which are the most frequent ones for the remaining 3 countries. This analysis directly addresses RQ1 by revealing patterns in coding approaches across regions.

Beyond the technical differences, the clustering also reveals patterns in code organization, variable naming, and commenting styles. The European solutions generally demonstrate more consistent patterns which could be due to standardized educational or professional practices. Rwandan solutions show higher variability, with some matching European standards and others displaying more unique approaches.

This analysis supports the findings of the interviews which is that educational background significantly influences coding style, with more standardized European educational systems producing more consistent approaches compared to the more diverse educational backgrounds of East African software engineers. During the interviews, this was noted by almost all Rwandan engineers, as well as some Europeans stating that their education is uniform across the country.

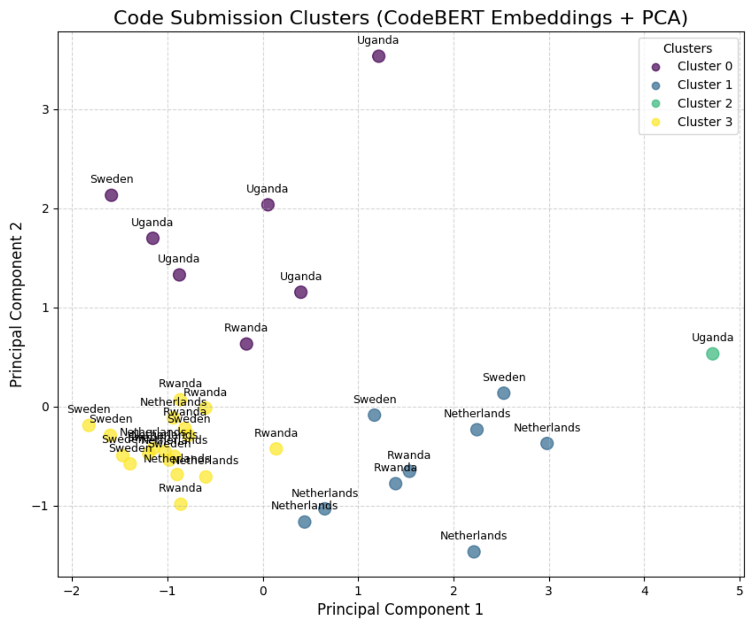

Figure 4.1: Code Submission Clusters using codeBERT embeddings projected via PCA.

Furthermore, the clusters can be represented in another way using a barchart as shown in Figure 4.2

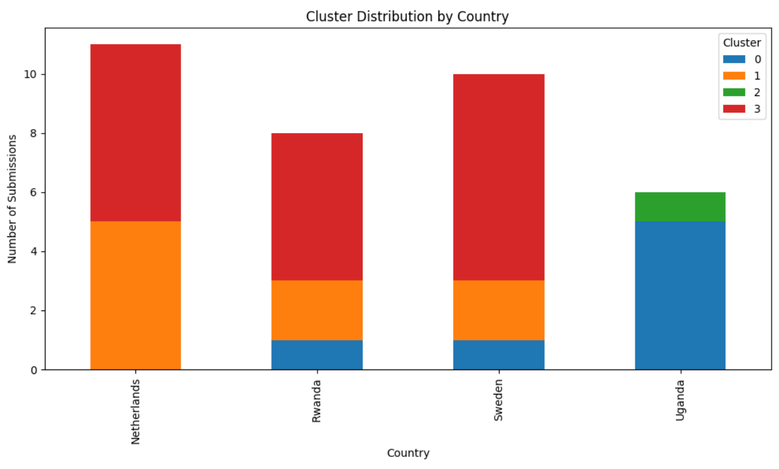

Figure 4.2: Code Submission Clusters using CodeBERT embeddings shown through bar graph.

The barchart in figure 4.2 provide a complementary visualization of the clustering analysis. It is showing the distribution of ways to approach the solution across the countries. It is obvious that Swedish and Dutch engineers show similar patterns in their code (Most results in clusters 1 and 3), but also Rwandan engineers showing a surprising similarity to their European counterparts. In contrast, the valid submissions of the Ugandan are concentrated in clusters 2 and 4 which suggest a different approach to problem-solving. This addresses RQ1 by quantifying the similarities and differences in coding patterns between regions, and shows that Rwandan code patterns in particular have a striking similarity to European coding patterns.

4.1.2 System Design Challenge

In the system design challenge The Netherlands demonstrated the most overall technical maturity across all experience levels. Their software engineers delivered consistent quality responses that reflected a good educational foundation and professional training. They were systematic about problem solving with balanced attention to security, performance, and operations. The small difference between junior and senior skill levels could be reflective of good knowledge transfer within their development community, which was also commented upon during the interviews. Dutch engineers all stressed security and monitoring, preferred PostgreSQL together with Redis with advanced caching, and showed better Development and Operations (DevOps) maturity with Continuous Integration/Continuous Deployment (CI/CD) and container orchestration knowledge. They had a professional and structured tone of communication independent of experience, with good reasoning for technical decisions.

The Swedish engineers showed a strong technical foundation, with a clear progression going from junior levels to senior levels. The knowledge the software engineers had were consistent with their experience. They focused more on performance optimization and caching strategies. Most often they preferred PostgreSQL + Redis like their Dutch counterparts, but they demonstrated more variation in implementation details. Swedish software engineers offered practical approaches focused on core requirements where they tried balancing theory with implementation concerns. Their communication was generally clear, but where confidence in answers increased together with experience level.

The Rwandan software engineers showed the highest variation in technical skills, where there were some exceptional software engineers across all experience levels together with more basic responses. They showed the most diverse approaches, including serverless and document storage architectures, and they also mentioned a wider variety of technologies with less standardization. Their communication and problem solving approaches ranged from very low, where some participants could not answer the design question at all, to some few very comprehensive answers. They had less predictable progression patterns across experience levels, as there were instances of seniors not being able to answer the question, but also some juniors who were on par with European senior engineers in their answers. The reason to this could be the uneven access to both education and quality work experience present in the country, which also was noted by Rwandan engineers with a higher education and experience working in large software companies. This however highlights the potential in the individual, as this was evident in standout performances from both junior and senior engineers. The interviews also revealed that some approaches studied were never put into practice before, but that due to their own enthusiasm, the engineers had looked into the theory behind architectural design previously, and could draw on that rather than formal education or work experience.

Overall, Dutch seniors were consistent with giving comprehensive technical approaches and had a strong emphasis on modern cloud architecture, security, and monitoring. They showed a very high DevOps maturity and communicated with a confident tone. Swedish seniors had a strong technical foundation but that somehow varied from each other. They had a good understanding of service separation and caching strategies. They also had mixed communication styles that ranged from detailed step-by-step explanations to more conceptual approaches. Rwandan seniors were the most diverse in their designs, and presented different approaches varying in completeness. It ranged from specialized serverless architectures to document database solutions, to having one senior not being able to complete the challenge at all. They also had very variable communication styles.

In the junior population interviewed, the Dutch junior engineers showed strong awareness of security and a good architectural understanding. Their responses consistently had PostgreSQL and Redis with caching strategies and many of them also mentioned containerization and monitoring concepts that would normally be expected of a senior engineers. Swedish junior engineers showed good foundational knowledge with that corresponded with their experience. In general they understood basic architectural concepts and made reasonable picks of technology used. Their explanations were concise and technical, and they covered important concepts like key-value databases and load balancing. Lastly, they showed good awareness of team context and practical implementation concerns, which could suggest good industry exposure despite only a few years of experience. Lastly, Rwandan juniors were the most variable group. Several of them provided minimal or no responses to the system design question, or were not even able to solve it, while they also had some standout performers that gave comprehensive responses with detailed Application Programming Interface (API) designs, encoding explanations, and database schemas. This extreme variation is likely due to the significant differences in educational background and learning opportunities among the Rwandans interviewed, since some juniors had already been able to work at Rwandan IT consultancies and acquired a masters, while other lower performers did not have this extensive background.

When looking at technology preferences, we can see different ecosystem differences. This is as the Dutch consistently referenced complete modern stacks, while Swedish engineers showed strong cloud-native awareness, and lastly Rwandan engineers demonstrating more varied technology familiarity.

The gap between Dutch juniors and seniors was smaller than expected, while Swedish engineers showed the clearest progression by experience level. Rwandan engineers had the most individual variation, making group generalizations less reliable. The most consistent good quality responses across all experience levels came from Dutch engineers, followed by Swedish engineers. Ugandan engineers were not taken into account in this part of the study, as all but 2 software engineers did cheat on this part, and the rest failed to provide an answer.

4.1.3 Code Review Analysis

The code review challenge in Section 3.3 looked at the sentiment of developer comments, which are showcased in Table 4.2. The table shows both cultural and communication differences in feedback styles. Despite high neutrality across all countries, compound sentiment scores indicate clear contrasts.

Rwandan and Ugandan reviews had a more positive tone in their comments, with compound sentiment scores close to +1, which suggests more encouraging or supportive feedback styles. On the other hand, Dutch reviews were overall having a negative tone, while Swedish engineers had a neutral tone. During the interviews, software engineers from NWE answered that their tones do not imply hostility but rather critical or direct feedback styles which they themselves preferred, and that they believed would be appreciated by their peers and that this was more common in European engineering cultures.

These differences do have implications for a cross cultural team, because while Rwandan and Ugandan teams may focus more on emotionally supportive environments, Swedish and Dutch teams might be more task-oriented and direct, which could potentially result in conflicts.

Table 4.2: Sentiment Analysis Results of Code Reviews by Country

Country	**Neg-ative**	**Neut-ral**	**Pos-itive**	**Com-pound**	**Overall Sentiment**
Rwanda	0.049	0.818	0.133	0.541	Positive
Sweden	0.113	0.780	0.107	0.007	Neutral
Netherlands	0.153	0.764	0.083	-0.361	Negative
Uganda	0.047	0.713	0.240	0.919	Positive

4.1.4 Security Knowledge Analysis

As seen in Figure 4.3 it is clear that the identification rates for security issues differ across different engineering groups. Ugandan engineers will not be showcased, as all but 3 participants were caught using ai on this part of the examination.

Figure 4.3 shows that Rwandan software engineers were more varied in the security issues they found, while Dutch and Swedish software engineers focused on the top end of the spectrum. Surprisingly, Dutch juniors underperformed on this task compared to their previous performances, even though they showed strong security understanding in the challenge in Section 4.1.2. The interviews failed to uncover the reason for this.

The pattern of security issues identified by junior engineers in (Figure 4.4) shows differences in awareness levels between countries. Swedish junior engineers showed highest awareness, followed by juniors from the Netherlands, and at last Rwandan juniors with the lowest identification rates.

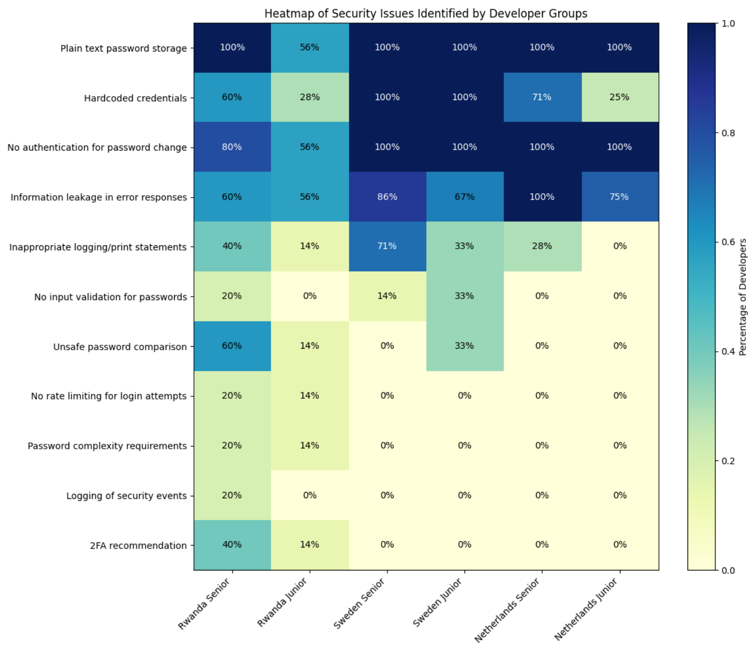

Figure 4.3: Heatmap visualization of security issue identification rates across all software engineering groups.

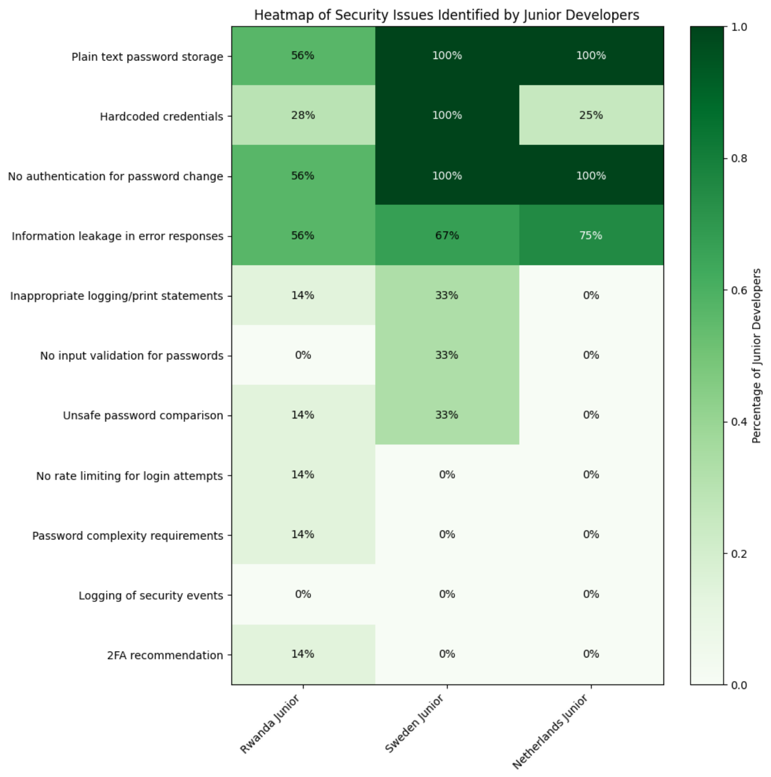

Figure 4.4: Heatmap visualization of security issues identified by junior software engineers across different countries.

The recognition of security issues among senior software engineers in Figure 4.5 vary in what is identified by each region. Senior Swedish and Dutch software engineers focus on plain text password storage, hardcoded credentials, authentication for password change and information leakage, while the Rwandan engineers are more versatile and showing knowledge scattered around many different areas.

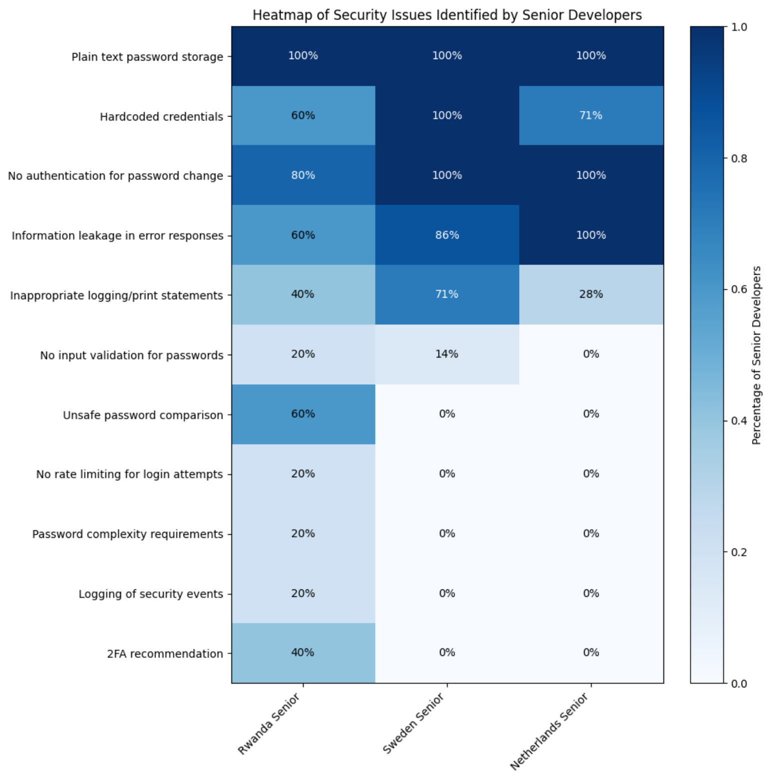

Figure 4.5: Heatmap visualization of security issues identified by senior software engineers across different countries.

Figure 4.6 visualizes the differences in skills between seniors and juniors across the regions. It shows that senior software engineers outperform juniors in identifying security issues across all countries. However, Seniors in Rwanda show a similar level of knowledge as Swedish ones, and outperforming Dutch seniors, which is surprising as compared to the results from the challenge in Section 4.1.2. When asked about why the results were so, Rwandan team leads speculated that it might be due to the hours of work they put in, and the exponential knowledge increase they see when growing in seniority. When European team leads were queried about this, they also speculated that Rwandan seniors amass more tech-work hours across their career, and one noted that work there is not only work, but survival, while work in Europe is inherently different. Lastly, Rwandan juniors are the weakest performers, falling slightly short of Dutch juniors and performing much worse than Swedish ones. This was once again speculated by team leads from all countries to be because of limited exposure to projects and a lower quality education.

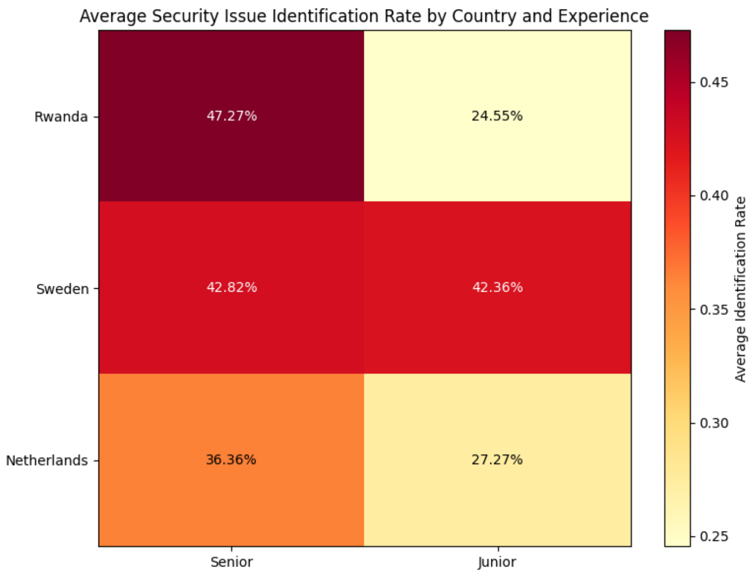

Figure 4.6: Heatmap comparison of compounded security awareness between senior and junior software engineers across Rwanda, Sweden, and Netherlands.

The results from the technical assessments directly address RQ1 by showing patterns in the software engineering skills between the countries and regions. The clustering analysis shows that European software engineers employ more consistent algorithmic approaches, with Dutch and Swedish software engineers favoring optimal hashmaps and displaying stronger security awareness. However, in security issues identified shown in figure 4.6, it is shown how the difference between Rwandan seniors and juniors is very large, and that the seniors from Rwanda outperform their European counterparts, while the Rwandan juniors are outperformed by every other group, which is speculated to be due to the longer hours Rwandan software engineers work noted in the interviews, and in turn being exposed to more projects and having more effective hours of software engineering. Lastly, Rwandan software engineers showed higher variability in their technical approaches, where some had outstanding performances matching European standards but others showing more basic implementations or failing to implement a solution overall. These differences appear rooted in educational backgrounds and early career opportunities, as revealed in our interviews where several Rwandan software engineers mentioned limited access to standardized computer science education. The standout performance of some EA software engineers despite these limitations mean that there is individual potential that may be unrealized because of systemic factors rather than capability factors.

4.2 Biases and Process analysis

4.2.1 Process Steps

In this section the process steps from Section 3.3 will be presented. In the Ugandan group, interviews revealed a persistent English-as-a-second-language struggle. The inflated times are speculated to be a communication problem rather than process inefficiency, and therefore Uganda is treated as an outlier for the remainder of the process-duration analysis. Their data is still shown for transparency, but all cross-country comparisons are restricted to Rwanda, the Netherlands, and Sweden.

The table is divided into different process steps, where each process is broken down into a step. The software engineers are told to give a time estimate for how long each step takes to complete.

Process	Step	RW	NL	SE	UG
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
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
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

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

Table 4.3: Self-reported software development process time (minutes) estimates by country

4.2.2 Longest vs Shortest Process Steps per Country

As shown in figure 4.7, the Rwandan engineers are quick at bug fixes and documenting them, as well as going through their deployment process at a fast rate. Their slowest steps include the process of development, writing tests and doing their post deployment checks.

Figure 4.8 shows that the Dutch software engineers spend the least time running their deployment tests, doing post deployment checks, and waiting for a final approval. Their slowest steps include planing how to solve an issue, fixing bugs and the process of development.

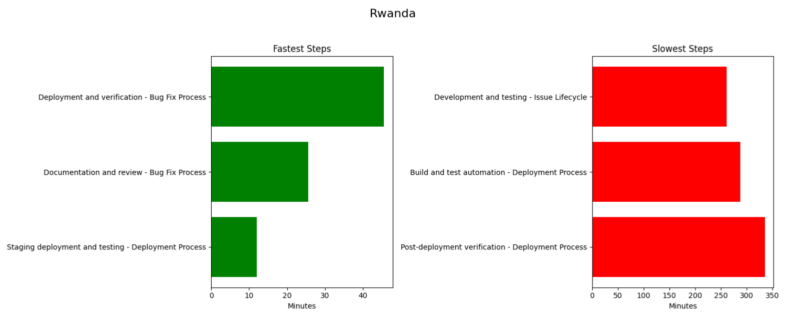

Figure 4.7: Steps that take the longest as well as shortest for Rwandan software engineers.

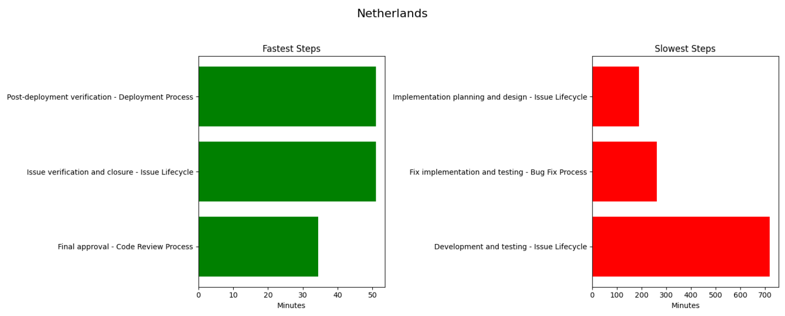

Figure 4.8: Steps that take the longest as well as shortest for Dutch software engineers.

Figure 4.9 shows that the Swedish software engineers spend the least time running their tests when doing their code review, documenting bug fixes and approval in the code review. What takes the most time for Swedish software engineers is creating the build and test automation in the deployment process, planing how to solve an issue, and lastly the process of development for an issue.

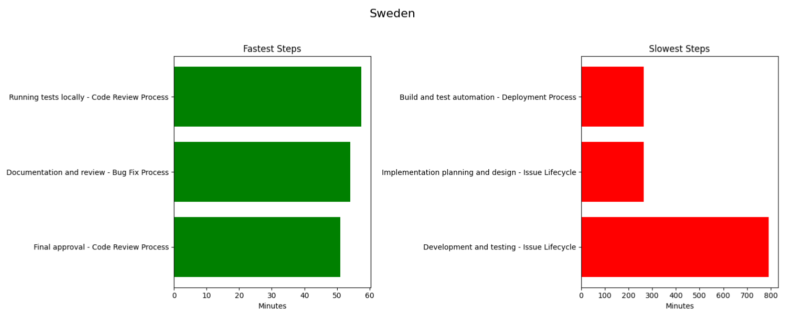

Figure 4.9: Steps that take the longest as well as shortest for Swedish software engineers.

Figure 4.10 shows Ugandan software engineers fastest and slowest times.

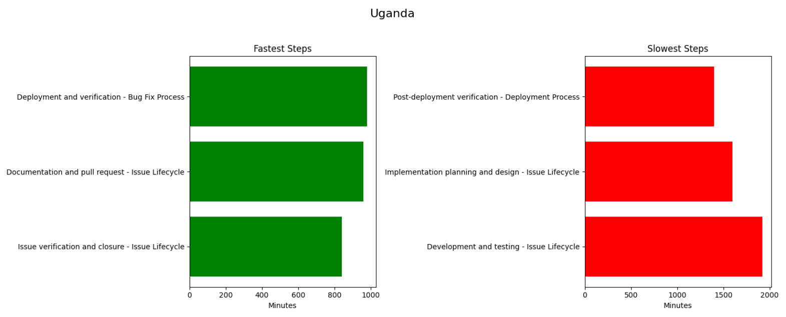

Figure 4.10: Steps that take the longest as well as shortest for Ugandan software engineers.

Critical path analysis

Table 4.4 visualizes the process steps that take the most time in each country. It is seen that the Dutch and Swedish software engineers focus the most on development and testing, while the Rwandan software engineers instead put in a third of the time here, and instead focus more on the deployment phase of their project.

Table 4.4: 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%

4.2.3 Biases

In this section the results from the bias questions from Section 3.3 will be presented, together with relevant information mentioned by the software engineers regarding biases.

Figure 4.11 shows that Rwandan software engineers see themselves mostly as equals to their European counterparts, but as slightly better than Ugandan software engineers. In their rating, they consistently mentioned that ”Ugandan developers are good”, but yet ranked them lower than those from NWE and themselves. They also believed themselves as individuals to be at least on par with Europeans in skills, but that the average Rwandan lacks in comparison to NWE. Some Rwandan engineers had worked with Dutch developers, and therefore ranked their communication skills very high. In general, they saw Swedish engineers as entrepreneurial, and Dutch ones as hardworking.

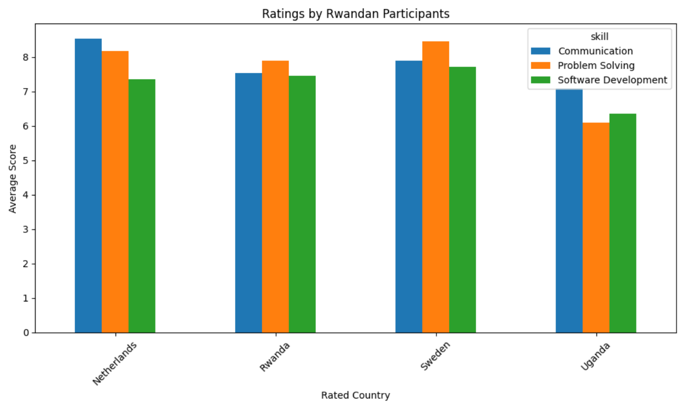

Figure 4.11: Perception-based given by Rwandan participants to developers from Rwanda, Sweden, Netherlands, and Uganda.

Swedish participants rate themselves the highest, followed by Dutch software engineers. Rwandan and Ugandan software engineers both received low scores across all three categories. However, the Swedish software engineers noted that they believe these software engineers to be capable and have good problem solving and communication abilities, but that to their understanding, the lack of technology and education about software engineering make them poor coders. The Swedish software engineers made no distinction between Uganda and Rwanda, and saw no difference in them as software engineers.

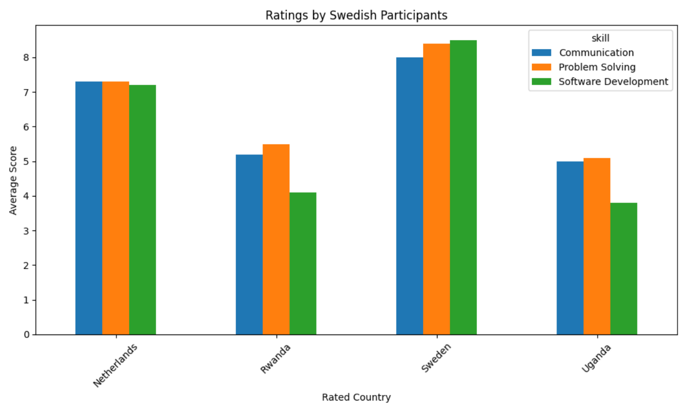

Figure 4.12: Perception-based given by Swedish participants to software engineers from Rwanda, Sweden, Netherlands, and Uganda.

Similar to Figure 4.12, it is seen that Dutch software engineers rank themselves slightly higher than Swedish software engineers, as seen in Figure 4.13. They see Rwandan and Ugandan software engineers as much less capable than European ones. When queried about it about half said that they do not know enough about the countries, and that they put them under the umbrella of Africa, and that because of that the deduce that they do not have enough experience or knowledge. The other half were more confident that these were underdeveloped African countries, and that they are never in the discussion of software engineering in Africa like other countries, such as Kenya.

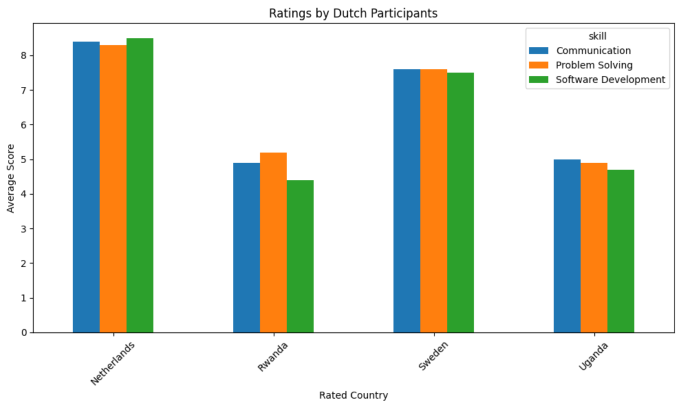

Figure 4.13: Perception-based given by Dutch participants to software engineers from Rwanda, Sweden, Netherlands, and Uganda.

Ugandan participants rated themselves slightly above Rwandan ones, but still see Dutch and Swedish software engineers as more capable than themselves. They were very confident in Dutch and Swedish software engineers being better due to the opportunities NWE presents in terms of education, infrastructure and job opportunities. They also deemed themselves to be better than Rwandans, and believed Rwanda to be lacking in digital infrastructure and good software engineers.

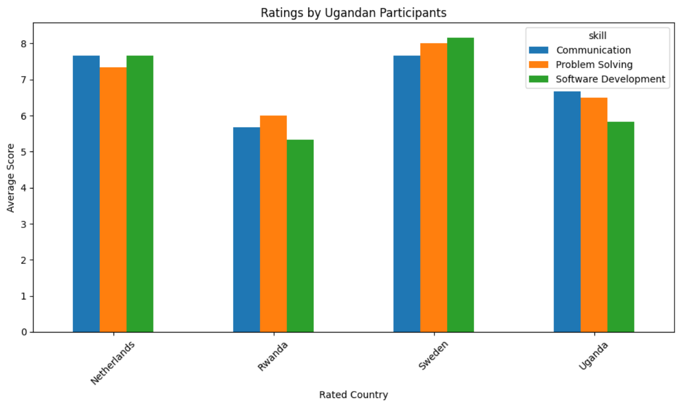

Figure 4.14: Perception-based given by Ugandan participants to software engineers from Rwanda, Sweden, Netherlands, and Uganda.

On average, it is seen in Figure 4.15 that across all developer groups, the European ones rank the highest as compared to Rwandan and Ugandan ones. The interviews with Rwandan team leads showed that they were not surprised by the results, as they expected to be subject to such biases. Swedish and Dutch team leads were not surprised either, and believed that this was reflective of the true skill distribution across the countries.

The perception data reveals biases that directly address RQ3. The gap between how software engineers from NWE rate EA software engineers (consistently below 5 on a 10-point scale) and measured performance indicates potential perception biases. These biases could be an issue for effective collaboration as it could prevent the recognition of complementary strengths across the regions. EA software engineers relatively balanced perception of software engineers from all regions suggests a more understanding view of global capabilities that could in turn help with cross-cultural teamwork. However, the internalized lower self-assessment from Ugandan software engineers compared to European software engineers also indicates how global perception hierarchies may affect confidence and how each developer presents themselves in an international context. During the interviews, high performing Rwandan software engineers said that Rwandan developers as a whole are worse than European ones, but deemed themselves to be on the same level, contrary to Ugandan ones, which saw both themselves and their peers as less capable.

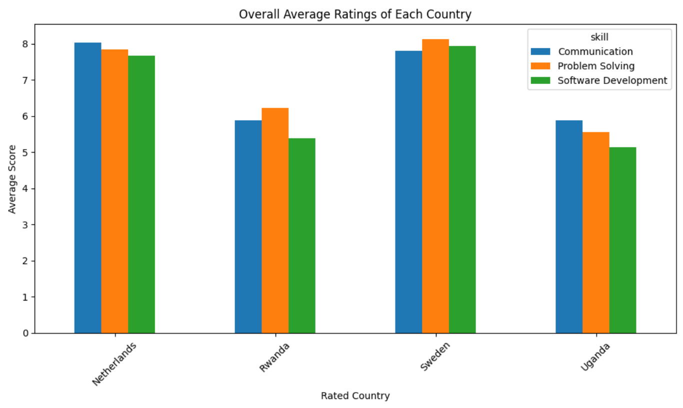

Figure 4.15: Average ratings per country

4.2.4 Process Step Visualization

Figure 4.16 shows the software development process flow, where each circle represents a process step. The larger the circle is, and the more red the color is, the more time is spent on that process. The flow starts with the Issue Lifecycle, followed by Code Reviews, Deployment Process, and lastly Bug Fixes. The figure shows that Swedish software engineers spend the most time on the issue lifecycle, where they plan and develop their given programming tasks. As the most time is spent here, much less time is needed for code reviews, which is the fastest process step. After this, the deployment process goes quickly as well. Lastly, bug fixes take more time than code reviews, but still are done pretty fast considering the comparison to the issue development stage.

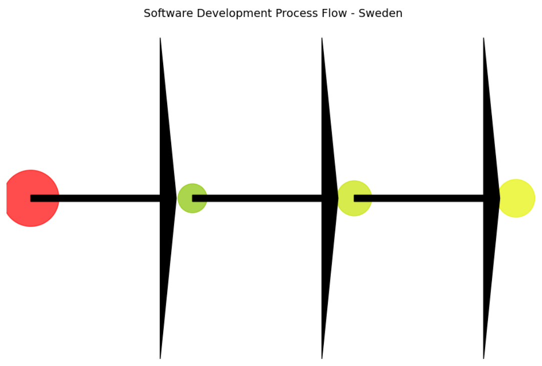

Figure 4.16: Process visualization for Sweden

Figure 4.17 shows that Rwandan software engineers spend the most time on the issue lifecycle, as well as the third step, which is working on their DevOps pipeline and deployment process. It is also seen that the second step takes some time, which is the process of code reviews. Lastly, bug fixing, which is the fourth step, goes extremely fast, due to putting time and effort into having a strong DevOps pipeline, good code reviews, and spending much time developing a robust solution.

Figure 4.18 shows that Dutch software engineers spend the most time on the issue lifecycle part of development. The code review process is the shortest step, followed by the deployment process. Lastly, the second longest time is put on fixing bugs. The process mimics the one of Swedish software engineers, where not much time is put into code reviews and the DevOps pipeline, and instead more time on fixing the inevitable bugs that will appear.

Figure 4.19 shows the process of Ugandan software engineers. As mentioned prior, the hurdle of language caused them to spend lots of time on every step, and this can not be seen as valid data.

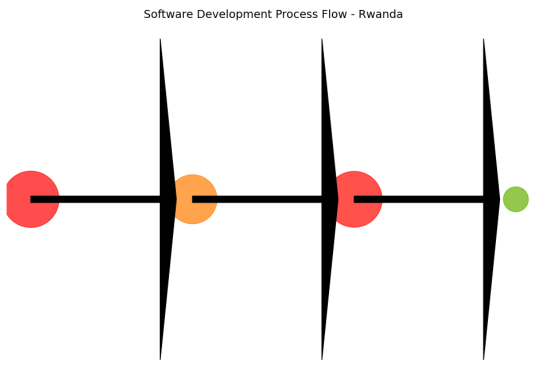

Figure 4.17: Process visualization for Rwanda

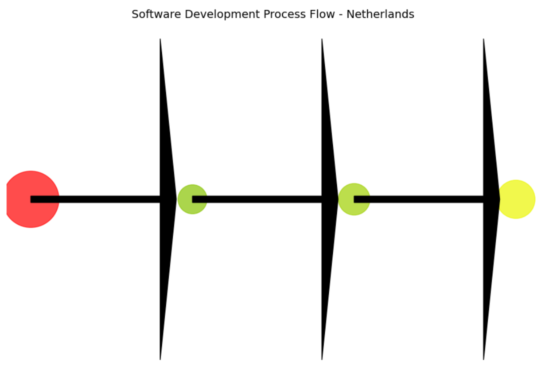

Figure 4.18: Process visualization for the Netherlands

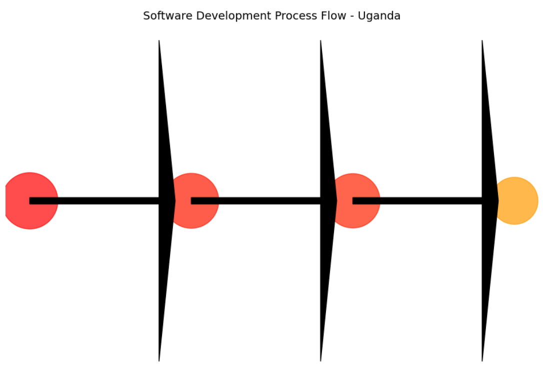

Figure 4.19: Process visualization for Uganda

The quantitative data serves to support and contextualize the qualitative interview findings rather than standing alone as definitive statistical evidence. For instance, the process time differences across countries provide quantitative support for qualitative observations about different development approaches.

The process analysis and communication style findings directly answer RQ2 by showing differences in software development approaches across regions. The sentiment analysis in Table 4.2 in Section 4.1.3 shows that software engineers from EA employ more positive, encouraging feedback styles (compound scores >0.5) compared to the more critical, direct approach of European software engineers. The differences in communication also extend to the process emphases, where European teams put more time into initial development while EA teams put more emphasis on code reviews, bug fixing and and the deployment process.

These patterns reflect broader cultural approaches to problem solving, as the Rwandan team leads mentioned collaborative verification and constructive feedback, while their European counterparts favored individual development time and direct critique. These differences in communication and process priorities answer RQ2, and will be further discussed in the Discussion chapter, as it could have important implications for cross-cultural software teams.

4.2.5 Results summary

The analysis reveals patterns of similarities and differences that address our three research questions:

RQ1: software engineers from NWE showed more consistency in algorithmic approaches and system design, with Dutch engineers showing strength in security awareness and Swedish engineers showing clear skill progression from juniors to seniors. Developers from EA, and particularly those from Rwanda, displayed a higher variability with both high performers and basic implementations, where some outstanding participants were on the same level of their European counterparts.

RQ2: Communication styles varied a lot, with software engineers from EA using a more positive and encouraging feedback (a compound sentiment scores >0.5) while Dutch software engineers had a more critical and direct approach (a compound score -0.361). These communication differences extended to what is emphasized more in the software development process, where European teams allocate more time to initial development and EA teams focusing more on verification and the deployment process.

RQ3: Large perception gaps exist where European software engineers are underrating EA software engineers capabilities compared to both self-assessments and measured performance in the study. These biases could impede effective collaboration by preventing recognition of complementary strengths across regions. However, European software engineers do recognize that this is most likely due to weak technical education and infrastructure in the country, and nothing to do with the individual themselves.

4.3 Reliability Analysis

The assessments were standardized across all locations. However, the interviews with Dutch software engineers were done online, but with a second person present in the room to avoid cheating. Ugandan software engineers were incentivized to join the assessment where the top performers would be given a job interview as a software engineer at a Swedish outsourcing company, which could be too much incentive to do good and thereby making the software engineers cheat.

All software engineers choose their own time slots, while Ugandan ones were given time slots in the morning. Many of them also had to travel long distances to reach the assessment area, which in turn could have lead to fatigue. Furthermore, misinterpretations in the assessment goals and tasks were noted at every Ugandan interview, despite English being a national language.

4.4 Validity Analysis

To enhance the statistical validity of the clustering results, a bootstrap resampling analysis with 1000 iterations on the code submissions was done, which addresses validity concerns present in the dataset [104].

The bootstrap analysis systematically evaluates cluster stability by repeatedly sampling the dataset with replacement. It performs k-means clustering on each resampled dataset, and tracks the frequency of pairs of code submissions that appear in the same cluster. In this way discrete cluster assignments are transformed into probability-based relationships, providing a better understanding of code similarity patterns [105].

A silhouette score of 0.3301 was given by the initial clustering, which indicates moderate cluster separation. This suggests that meaningful patterns exist in the data, but with overlap between clusters. In general, traditional clustering approaches might be insufficient when looking at the moderate silhouette score, as cluster boundaries are not strongly defined [106, 107]. The bootstrap method however overcomes this limitation by quantifying the consistency of relationships between code submissions across 1000 random resamples [108].

The perception bias data shows validity through three measures. To begin it has an inter rater reliability score (0.846), which shows high consistency among participants when rating the same targets and indicates that it is real perception patterns that are being measured rather than random variations [109]. Furthermore, the sample has sufficient statistical power to detect the effects present in our data, the lowest detectable effect size (Cohen's d) was calculated at 0.6677, which was below the actual largest effect size of 2.89 between how software engineers from NWE view themselves versus EA [110, 111]. Finally, as emphasized in mixed-methods research literature, the quantitative findings are strengthened through triangulation with qualitative insights. This gives deeper contextual understanding of the perception biases identified in the statistical analysis [112].

However, with the sample size of 48 software engineers across four countries, statistical tests have limited power. The analysis for the most part relies on statistics that is descriptive, which is also supported by qualitative insights rather than inferential statistics alone. As mentioned throughout the study, this mixed-methods approach provides a more comprehensive understanding than statistical significance testing could offer with the sample size of the study.

A statistical test was made to assess the trend of Rwandan seniors outperforming their European counterparts. This was done through a Wilcoxon signed-rank test which compared per-issue identification rates [113]. The results were not statistically significant at a 5% level, with W = 36.0 and p = 0.193. However, the qualitative interview data reinforced the idea that Rwandan senior software engineers had greater practical exposure to a wide range of security scenarios as they often worked on many live deployments, which may contribute to their strong performance. Therefore, although the statistical evidence is insufficient to confirm a significant difference, the qualitative context indicates that the claim is still plausible.

Selection bias remains a limitation as participants were recruited through professional networks rather than random sampling. Swedish and Dutch participants represent accomplished engineers and high achievers from top universities in each respective country, working at highly funded VC startups and large enterprises. Rwandan engineers were split between two high achieving IT consultancies, and then random engineers willing to do the assessment, some being free lancers and some working at smaller Rwandan tech companies. Ugandan software engineers looking for a job were provided through a third-party. As noted by Baltes and Ralph, volunteer bias in software engineering studies can affect results by over representing engineers with higher confidence in their abilities [114].

Interviewer bias was mitigated by using standardized assessments and evaluation criteria. The same interviewer conducted all sessions to maintain consistency.

Communication styles in code reviews showed clear regional patterns, with sentiment analysis revealing more positive feedback approaches in EA contexts compared to more direct critical feedback in NWE contexts. These differences align with Hofstede's cultural dimensions regarding power distance and individualism/collectivism [33].

Self reported time estimates for development processes could be influenced by cultural tendencies toward optimism or conservatism in estimation. The way different cultures do time estimations varies across the cultural contexts as some cultures systematically underestimate or overestimate required effort [115]. Still, this can not give a overview of how long different processes take, but instead shows what processes are deemed the hardest or most important to put time on across each country.

Chapter 5

Discussion

This chapter discusses the key findings from the comparative study and the implications of the findings for GSD. It explores the technical differences between regions and focuses on how systemic educational advantages account for performance differences, rather than inherent capabilities. The discussion examines how cultural dimensions such as power distance and individualism/collectivism influence communication styles. It also looks at process differences as alternative optimization strategies adapted to different contexts. Then the perception gaps found in the study are discussed as potential barriers to effective cross-cultural collaboration. Finally, methodological reflections evaluate the strengths and limitations of the pragmatic mixed-methods approach, suggesting improvements for future research in cross-cultural software engineering studies.

Technical Skills and Implications

The technical assessments in the study show the differences in consistency across the regions. Developers from NWE showed high consistency across all assessments, while Rwandan software engineers showed some software engineers to be on the same level of their European counterparts and some drastically under performing. This is assumed to be because of two reasons, the first reason being that the European developers are all highly educated and work at large enterprise companies or VC funded startups. The second reason, which was noted in the interviews, were that the Rwandan software engineers that received good results on the technical assessments were also software engineers who worked at reputable Rwandan IT consultancies, having gained exposure to working with international companies, as well as having masters degrees from decent schools and some even contributing to open source projects during their free time to further develop their skills. The low performers usually were seeking jobs while freelancing, or often juniors that are early in their careers. There was also one instances of a developer in reputable consultancy in Rwanda that did not perform well. Another insight from the interviews were that Rwandan seniors worked many more hours a week than their counterparts from NWE. Time ranges around 60 hours peer week and many concurrent projects were not unusual for Rwandan seniors, while seniors from NWE noted that they very often worked less than 40 hours a week. This could be the reason for why some seniors, despite not having the same quality of education as European ones, could amass the knowledge by over time working many more hours.

The clustering analysis using codeBERT embeddings also confirms the hypothesis of different strengths in educational backgrounds as it shows clear patterns in approaches taken. European solutions generally demonstrated more consistent patterns which is likely due to standardized educational practices, as opposed to Rwandan solutions that showed higher variability. Interestingly, some Rwandan software engineers clustered closely with European developers, which is speculated to be due to the experience gained in international teams, but also possibly as noted by Rwandans themselves due to the establishment of Carnegie Mellon University Africa, where some of the high performers had attended.

This also brings us to the relatively surprising discovery of Rwandan seniors outperforming both Swedish and Dutch software engineers in finding security issues, while Rwandan juniors fall behind their European counterparts. This can once again be due to the vast exposure to projects the Rwandan seniors work with. Furthermore, Rwandan team leads noted that their teams work on multiple projects together, and often consult each other on various topics which in turn leads to much knowledge sharing. When queried about their security knowledge, Rwandan software engineers noted this to be the highest, while the AI knowledge in their teams seemed to be lower compared to European ones, though they were also trying to establish at least a baseline by allowing some of their software engineers to work on side projects related to AI.

These technical disparities have implications for cross-regional collaboration. The Rwandan standout performers show that organizations should recognize the high variance present in the region, rather than viewing software engineers from EA as uniformly less skilled. The performance of senior Rwandan software engineers suggests that with appropriate experience, initial educational gaps and smaller experience gaps can be overcome. However, Rwandan juniors remain harder to employ, but this is most likely to change due to the increased focus on fostering IT knowledge in the country [11]. Unfortunately, the data can not speak to the skills of Ugandan software engineers, and the validity of the data is not strong enough to make any claims due to the language barriers present during the interviews, and because of the use of unauthorized tools during the interview such as generative AI.

Systemic Factors vs. Individual Potential

The findings strongly suggest that the performance differences found in the study primarily come from systemic factors rather than inherent capability limitations. The standout performance of some software engineers from EA, despite significant educational disadvantages compared to NWE, indicates individual potential that may remain unrealized due to systemic barriers. Another note taken during the interviews is how eager software engineers from EA are to learn and showcase their skills, while software engineers from NWE were less enthusiastic about upskilling on their free time.

Several key systemic factors showed from the interviews and assessments. First, software engineers from every country spoke of the inconsistent quality of computer science education in EA. Particularly Rwandans and Ugandans spoke of it creating an uneven foundation for early career development for juniors. It was also noted during the interviews how some juniors did not have a higher than basic knowledge of data structures at all, even though they had completed a bachelors and started working as software engineers. When team leads were asked about this, they said that they themselves also had to upskill themselves after studying. The reason for this is most likely the low quality of education given at most EA universities, where only a few universities from Rwanda and Uganda are ranked between 1000-1500 in the world, with the others falling outside of the ranking system, compared to Swedish and Dutch universities that consistently are in the top 100-150 in the topic of Computer Science [116]. However, once again, the enthusiasm of these students is not to be taken lightly, as they are continuously upskilling themselves and seeking new opportunities, which is a great characteristic if wanting to hire based on potential [117]. Also, despite infrastructure advances, loss of internet connectivity is a common occurrence, and was noted as by a developer from Uganda to be frustrating as it makes it harder to get ”deep work” done.

Sentiment Analysis and Cultural Impact

The sentiment analysis of code reviews showed differences in communication styles between countries and regions. Developers from EA generally gave more positive and supportive feedback, meanwhile developers from NWE had either a critical tone (Dutch) or neutral tone (Swedish). These differences reflect deeper cultural patterns that influence workplace communication, since when team leads were asked about their practices, the responses cross country were different. Dutch software engineers did not feel like they necessarily had a negative tone, but rather that if someone is doing something wrong, they would be doing them a disfavor of sugar coating it, and that they themselves appreciate straightforward feedback. Swedish software engineers did not have any thoughts regarding culture, but plainly claiming that this is the way they write and its the most straightforward way to communicate by just pointing out what to keep and what to change. Rwandan software engineers were also neutral to the tone of their comments, but also had engineers saying that they are in the same seat in having a disadvantageous view upon them from the outside world, and that encouragement is always appreciated. As opposed to Dutch software engineers, Ugandan ones preferred positive feedback that still showcases clearly what the issue at hand is.

Drawing on Hofstede's cultural dimensions framework, these communication differences may correspond to variations in power distance and individualism/collectivism. Studies show that African countries generally score higher on power distance and collectivism [118], explaining their more supportive approach that values group harmony and positive reinforcement within hierarchical structures. This can also be a reason for why junior software engineers from Rwanda were under performing compared to their European counterparts, as due to the power distance they might be scared to ask for help, and in turn aid knowledge transfer across seniority levels. This was however not noted during the interviews.

European countries are typically on the other end of the scale to African software engineers, and they show lower power distance and higher individualism [118], leading to a more ”matter-of-fact” or direct approach that prioritizes task completion and technical correctness over maintaining relationships. The lower power distance also allows for room to be curious and ask questions, which could be a reason to the linear rise in knowledge from junior to seniors levels seen in Swedish software engineers. The Dutch perspective shows a low-context communication style where clarity and honesty are valued above social cushioning which can be contrasted to the

Rwandan and Ugandan software engineers comments that suggest awareness of different cultural expectations and the value placed on relationship-focused communication in their culture. These patterns should be taken into account when constructing GSD teams, so that misinterpretations of feedback style does not lead to any form of tension or misunderstanding in case members of the team are not aware of these cultural dimensions that potentially could affect communication. The high power distance is also noted by a Swedish team lead that has outsourced to EA previously, where his experience of software engineers from EA is that they are knowledgeable, but would rather claim sickness than ask for more time to complete a task, which was frustrating to him and in the end the reason for him terminating the collaboration.

Process Differences and Efficiency

The analysis of development process times showed notable differences in how teams from different regions allocate their efforts. Teams from NWE focused more on the initial development and design stages where they were allocating approximately 30% of total process time to development and testing, while Rwandan teams emphasized deployment, code reviews, and verification. Ugandan software engineers time estimations can not be taken into consideration due to language barriers.

These process preferences likely reflect cultural approaches to software development. Once again, drawing on Hofstedes work, the individual parts of the software development cycle, which is solving an issue, are prioritized by Europeans, while code reviews, which can be argued is a more collaborative tasks, is prioritized in EA. When asked about their time approaches, both Dutch and Swedish software engineers noted that they work upfront on proper architecture and design to not have any issues later, very often referencing tech debt. Rwandan software engineers felt like development was often not what takes time, but rather code reviews, where many bugs were actually being addressed, leading to less time spent on fixing bugs. Furthermore, many Rwandan developers mentioned a rigorous DevOps pipeline, where they collaborated on creating many tests for their code, and in turn, having less bugs to solve alone.

Rather than representing efficiency variations, these differences can instead be seen as alternative optimization strategies adapted to different contexts. Neither approach is inherently better, as the Rwandan approach makes sense when you have more variability in individual skill levels as it catches issues through collaborative verification. These approaches can complement each other and have potential benefits in global teams. European software engineers developing robust code would be well combined with a highly developed DevOps pipeline as well as thorough code reviews. It is also important to note that developers from NWE showcased theoretical DevOps knowledge, but when questioned about the time estimates they believed that the importance of a robust testing suite is not always necessary, and that ”less is more” in these regards. Organizations could leverage these different strengths by structuring global teams where in a way where European software engineers lead initial architecture and design, while EA software engineers contribute to quality assurance, deployment automation, and post-deployment verification.

Perception Biases and Their Impact

The perception study revealed biases that potentially could affect cross-cultural collaboration. Developers from NWE consistently rated themselves and each other more highly than their counterparts from EA, despite performance data showing overlap and individual excellence from Rwandan software engineers. This is a perception gap which presents a substantial barrier to effective collaboration [37].

Rwandan software engineers tended to view themselves as equal to Europeans but superior to Ugandans, while Ugandan software engineers saw Europeans to be better than themselves across all aspects, but them being better than Rwandans. Many Rwandan software engineers commented that they believed that they would not be perceived in a good light by European software engineers. Rwandan software engineers were also sometimes the only Rwandans in international projects, and felt pressure to represent all Rwandan software engineers. Rwandan juniors many times commented on software engineers from NWE being better than developers from EA, while seniors from Rwanda noted that they are as good as any European developer, and that the difference is not ability but rather opportunity.

Developers from NWE many times said that they do not know a lot about the state of technology in EA, but that they believe it to be very low and undeveloped. During the interviews, the software engineers from NWE gave varying comments. Around half of the software engineers from NWE did not believe the developers to be very skilled, and some drew a parallel to working with outsourcing teams in Asia, where they have not had good experiences from, and in turn believed that EA teams would perform worse due to infrastructure and education barriers, and from this deducted that software engineers from EA would not be suitable to work with. The other half believed that there is skill to be found in EA on par with NWE, but that the ratio of good to bad software engineers is lower in EA than NWE.

Methodological Reflections

The pragmatic mixed-methods approach used in this study provided an extensive understanding of cross-cultural practices in software engineering and GSD. However, some of the methodological considerations do warrant a discussion. The combination of quantitative assessments and qualitative interviews made it so that it was possible to identify patterns while also exploring the contextual factors that shape those patterns, but the relatively small sample size (10-15 developers per country) limits the statistical power of the quantitative analyses, particularly at the country level, and even more when breaking it down to seniors and juniors.

The validity of the clustering analysis was strengthened through bootstrap resampling, which did confirm the stability of the patterns found despite having a moderate silhouette score of 0.3301. Also the perception bias findings demonstrated validity through high inter rater reliability (0.846) which indicates consistent patterns rather than random variation. Even though the validity is strengthened, findings should be interpreted as exploratory rather than definitive truths.

Selection bias remains a limitation, as participants were recruited through professional networks rather than random sampling. Rwandan participants were purely recruited from Kigali, and many of them from high end consultancies, rather than software engineers from smaller towns and rural areas. The European sample is also from top-teir companies and schools, which might not represent the average developer from Sweden and the Netherlands, but rather the highest performers. Ugandan software engineers were recruited from a forum, with an incentive of a small monetary reward and job interview opportunity if performing well, which in hindsight might have been the reason to the high number of cases of AI use, which was unauthorized.

The study aimed to collect data in place in all 4 countries, but this was failed to be achieved, as data from the Netherlands had to be collected online, as the author got sick after visiting Uganda and needed to go back to Sweden.

Future research methodologies in cross-cultural software engineering studies should consider larger and possibly longitudinal designs that capture development over time, as well as methods that control for selection bias. If research was to follow cohorts of software engineers over time, it would insights into how the gaps seen between juniors and seniors in Rwanda evolve with experience, as well as understanding what leads to Swedish software engineers having such a great career progression. Additionally, incorporating analysis of logs, rather than relying on self-reporting, would enhance validity, but would be out of the scope for this study. Self-reporting gives another angle that shows what each developer feels is the most important, and in turn can complement or be complemented by log analysis. Lastly, the research is grounded in pragmatism, as following a more theoretical approach would not be ideal for this kind of study, especially since studies on social issues normally require methodological flexibility [14–16].

Despite these limitations, the pragmatic mixed-methods approach provides valuable insights into the complex interplay of technical skills, communication styles, and organizational practices across regions. The findings contribute to a understanding of GSD dynamics in a totally neglected area, and highlights the importance of considering cultural context in software engineering research and practice.

Chapter 6

Conclusions and Future work

This chapter presents the conclusions of the comparative study of software engineers in EA and NWE. It also outlines limitations of the current research and identifies directions for future work that could build upon and extend this work.

6.1 Conclusions

The comparative analysis of software engineers from EA and NWE gave several significant findings that advance our understanding of GSD, specifically in EA. The research has successfully addressed the three research questions while revealing patterns in technical skills, communication styles, organizational practices, and perception biases across the regions.

Positive Effects and Outcomes

The most significant positive outcome of this research is the empirical demonstration that differences in software engineering practices across regions reflect complementary strengths instead of different capabilities and skill levels. The finding that senior Rwandan software engineers outperformed their European counterparts in security awareness challenges, as well as many standout performers from Rwanda being on parr with their European counterparts, contradicts assumptions about technical competency in emerging markets in EA, specifically in Rwanda. This insight aims to challenge the narrative that outsourcing to Africa involves a trade off between cost and quality, which can be noted due to the limited research on EA contexts in GSD and technical outsourcing.

The mixed-methods approach proved to be effective in capturing both the quantitative metrics of performance and the qualitative parts of the cultural context that was present. This methodology allowed for a more holistic understanding than each of the approaches alone could have provided. This was also especially true when it came to linking communication styles to cultural dimensions such as power distance and collectivism/individualism.

The research also successfully established a baseline for understanding cross-cultural software engineering dynamics in a highly understudied region. By focusing on Rwanda and Uganda rather than the more commonly researched African tech hubs like Kenya or South Africa (which themselves are also under researched), this work expands the geographical scope of GSD literature.

Evaluation of Results and Insights

The study successfully addressed all three research questions. RQ1 was addressed as the study identified clear patterns in coding skills and problem-solving strategies between regions. RQ2 was addressed as the study found distinct communication styles and organizational practices across cultural contexts. Lastly, RQ3 was addressed as it was revealed that there are significant biases in play that could affect cross-cultural collaboration.

Several insights were gained from the study. The most striking one was the disconnect between how software engineers from NWE perceived their EA counterparts and the actual measured performance. This perception gap represents a possible barrier to effective cross-cultural collaboration. Another insight was the standout performance of some software engineers from EA, where they despite educational disadvantages showed individual potential that remains unrealized due to systemic barriers. The out-performance of senior Rwandan software engineers in security knowledge, despite junior Rwandan software engineers lagging behind, shows how experience and dedicated practice can overcome initial educational and structural disadvantages. Furthermore, it was argued that the differences in process priorities reflect different optimization strategies in the software development lifecycle which are adapted to different contexts rather than reflecting efficiency variations. The Rwandan approach emphasizing collaborative verification makes sense when there is more variability in individual skill levels. Lastly, the sentiment analysis revealed how cultural contexts shape communication patterns in software engineering in the different regions.

Drawbacks and Limitations

Despite the positive outcomes, there are also several drawbacks that can be seen throughout the study. The small sample size (10-15 developers per country) limits the statistical significance of the findings, particularly when further dividing populations into junior and senior categories. While the qualitative insights help mitigate this limitation, broader generalizations should be approached cautiously.

The research also had challenges with data collection consistency, as interviews with Dutch software engineers had to be conducted online rather than in-person as initially planned, and as was done with all other countries.

Furthermore, the high invalidation rate for Ugandan submissions due to integrity concerns significantly limited the ability to draw conclusions about this population. The incentivization structure may have unintentionally encouraged unauthorized tool use. Due to budget and time constraints, the Ugandan population could not be retrialed, as the interviews are very long and need to be prepared well for, which showcases the challenges of empirical and in place cross-cultural research.

Lastly, selection bias remains to be a significant limitation as participants were recruited through professional networks rather than random sampling. The European sample represented all high achievers from top universities and companies, while the Rwandan sample was primarily from Kigali based consultancies with large international clients. This sampling approach may not represent the average developer experience in these countries, but instead their top performers.

Recommendations for Future Research

As mentioned in Section 6.1, this study faced several limitations that affected both the research process and the interpretation of results. For each limitation identified, I also propose alternative approaches that could address these constraints in future research.

Sample Size and Representation

The small sample size of 10-15 developers per country restricted the statistical power and generalizability of the quantitative findings. When further dividing these groups into junior and senior categories, it became too small for robust statistical inference. A future approach would be a larger scale study with at least 30-50 developers per country. This would provide more statistical reliability. However, it would also be very time consuming if trying to gain the same qualitative insights that this study did, but in turn strengthen generalization. Also, the sampling method that was used in the study also introduced a potential selection bias, as participants were high performers with strong backgrounds. It would be more advisable to implement a random sampling method rather than convenience sampling as this would enhance representativeness and data validity [87].

Data Collection Inconsistencies

The change from in person interviews to online interviews for Dutch participants was an inconsistency when compared to all other interviews that were made in place. The high invalidation rate for Ugandan submissions due to unauthorized tool use also worsened data quality for this population. A future approach would do all interviews in a similar manner. Additionally, the incentive structures would be either removed or remade to minimize motivation for unauthorized tool use. This would need to be further considered by future researchers.

Cultural and Language Barriers

Despite English being an official language in Uganda, there were clear language barriers present during the interviews with Ugandan software engineers. These communication challenges may have artificially affected performance metrics and limited qualitative insights. However, this only affected process data, as the if the developer has a hard time understanding the technical terms, they would not be considered a strong candidate to be part of a global team, and any mistakes they do on the technical assessments should be recorded. However, if this line of thought is not shared by future researchers, a possible approach would be to include local translators that could assist with the interviews when needed.

Scope and Methodology Constraints

The study focused on a limited set of technical skills and relied on self reported time estimates rather than analysis of actual activity logs. A future approach would include a broadened assessment to include additional dimensions of software engineering competency. To complement the self reporting, an analysis of actual project logs and repositories should be made. Ideally, a new study would implement a longitudinal component that tracks a subset of participants over a longer period of time to observe how skills develop, particularly comparing junior developer progression across regions.

Analytical Limitations

The moderate silhouette score in the clustering analysis and the use of sentiment analysis tools not specifically designed for technical communication introduced analytical constraints, as well as too small sample sized for generalization. A future approach would would employ more sophisticated analytical techniques such as combining multiple clustering algorithms and validating results through reviews by domain experts. For sentiment analysis, the development or adapting of tools specifically calibrated for technical communication in software engineering contexts could be developed, rather than VADER, which is based on informal communication [17]. They could also potentially incorporating domain specific lexicons for different cultural settings. Lastly, to quantify the data from technical assessments, each task could have been given a score that is then normalized over all countries to allow for a comparison, as well as comparing this to the bias data go give more tangible results to the study.

6.1.1 Future Work

Several areas of the study need further investigation, many of which were covered in Section 6.1. The most pressing next steps would include conducting a larger study with at least 30-50 developers per country using random sampling to strengthen statistical power and and allow for better generalization and representativeness. Additionally, as mentioned in Section 6.1, implementing a longitudinal component that tracks junior software engineers progression over time would provide insights into how initial skill gaps evolve with experience across the countries, especially because of Rwandas initiative to training 500 000 new developers by 2030, which is a large portion of their population [11].

The reliance on self reported time estimates rather than actual project logs work as a complement to real logs, however, no such study has been made. Future research should incorporate analysis of repositories, commit patterns, and collaboration logs to objectively measure productivity and code quality, and possibly utilize process mining techniques to uncover hidden processes. This would complement the interview based approach of this study and also eliminate potential biases in self reporting, as well as give view of how accurate the perception of each country is about their processes.

The lack of valid data from Ugandan software engineers shows a gap that should be addressed with better method approach. This could be including local translators and other, or no, incentive structures.

Finally, developing specialized sentiment analysis tools calibrated for technical communication in different cultural contexts would give a better understanding of how feedback is delivered and received across cultures.

6.2 Reflections

This comparative study of software engineers across EA and NWE has economic, social, and ethical implications. From an economic perspective, the findings challenge assumptions and prejudices about outsourcing to EA, and particularly to Rwanda, by demonstrating that technical competency is not uniformly lower than in NWE. Hopefully, this insight will help in potentially reshaping hiring practices in outsourcing companies and global companies and in turn open economic opportunities for both outsourcing companies and talented software engineers from EA, who have previously been ignored due to perception biases. At a larger scale, also stimulating GDP growth of EA.

In a social context the research shows how differences in education and infrastructure, rather than inherent capabilities, account for many of the seen differences in technical approaches. The variance among Rwandan software engineers, where some software engineers perform at levels similar to those of their European counterparts despite educational disadvantages, shows the importance of investment in technical education and infrastructure in emerging markets to further aid their software engineers, as well as looking at each individual as themselves and not a reflection of their country.

In an ethical context this work addresses inequalities in how developers from different regions are perceived and valued in the global marketplace. By visualizing the gap between perception and measured performance, the study challenges biases that may limit career opportunities for developers from EA. The findings on communication differences also raise important aspects of how cultural context influences workplace interactions and how awareness of these differences can lead to more inclusive collaboration.

The thesis contributes to the UN Sustainable Development Goals (SDG) number 8 (Decent Work and Economic Growth) and 9 (Industry, Innovation and Infrastructure) by providing empirical evidence that can inform more fair GSD practices and highlights the potential of tech talent from EA when given appropriate opportunities.

References

[1] L. Mann and M. Graham, *The internet and business process outsourcing in east africa: Value chains and networks of connectivity-based enterprises in kenya and rwanda*, https://eprints.lse.ac.uk/85053/, London School of Economics, 2014.

[2] M. Ghafoori, *Revolutionizing economic growth through ict: Rwanda's path to digital empowerment*, https://digitalcommons.bard.edu/senproj_s2024/106, Page 48, Bard College, 2024.

[3] M. Marinho, A. Luna, and S. Beecham, *Global software development: Practices for cultural differences*, https://arxiv.org/abs/1810.02350, arXiv preprint arXiv:1810.02350, 2018.

[4] S. Krishna, S. Sahay, and G. Walsham, "Managing cross-cultural issues in global software outsourcing," *Communications of the ACM*, vol. 47, no. 4, pp. 62–66, Apr. 2004.

[5] F. Merino, "Offshoring, outsourcing and the economic geography of europe," *Papers in Regional Science*, vol. 96, no. 2, pp. 299–324, 2017.

[6] Z. Iqbal and A. M. Dad, "Outsourcing: A review of trends, winners & losers and future directions," *International Journal of Business and Social Science*, vol. 4, no. 8, 2013.

[7] M. Tanner, "Communication and culture in global software development: The case of mauritius and south africa.," *Journal of Information, Information Technology & Organizations*, vol. 4, 2009.

[8] M. Tanner, "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, 2010, p. 388.

[9] E. E. Salami, C. O. Nwabuokei, and K. Ekundayo, “Software development and the security challenges in developing countries,” *Global Scientific Journal*, vol. 12, no. 12, pp. 927–941, 2024, ISSN: 2320-9186. [Online]. Available: https://www.globalscientificjournal.com.

[10] D. Govender. “The rise of african tech talent in global software outsourcing.” Accessed: May 2025, Scrums.com. (Sep. 2024), [Online]. Available: https://scrums.com/software-outsourcing/african-tech-talent-global-software-outsourcing.

[11] Taarifa Rwanda. “Rwanda launches ’1 million rwandan coders’ to build africa’s digital workforce.” Accessed: May 2025, Taarifa Rwanda. (2023), [Online]. Available: https://taarifa.rw/rwanda-launches-1-million-rwandan-coders-to-build-africas-digital-workforce/.

[12] V. O. Ajayi, *Primary sources of data and secondary sources of data*, Department of Science and Mathematics Education, Benue State University, Makurdi, Nigeria, Accessed 27 February 2025, 2016.

[13] O. A. Adeoye-Olatunde and N. L. Olenik, “Research and scholarly methods: Semi-structured interviews,” *Journal of the american college of clinical pharmacy*, vol. 4, no. 10, pp. 1358–1367, 2021.

[14] P. Feyerabend, *Against method: Outline of an anarchistic theory of knowledge*. Verso Books, 2020.

[15] J. Law, *After method: Mess in social science research*. Routledge, 2004.

[16] J. A. Maxwell, *A realist approach for qualitative research*. Sage, 2012.

[17] C. Hutto and E. Gilbert, “Vader: A parsimonious rule-based model for sentiment analysis of social media text,” in *Proceedings of the Eighth International AAAI Conference on Weblogs and Social Media*, 2014, pp. 216–225.

[18] H. Taherdoost, “Determining sample size; how to calculate survey sample size,” *International Journal of Economics and Management Systems*, vol. 2, 2017.

[19] J. G. Rivera-Ibarra, J. Rodríguez-Jacobo, and M. A. Serrano-Vargas, "Competency framework for software engineers," in *2010 23rd IEEE Conference on Software Engineering Education and Training*, 2010. DOI: 10.1109/CSEET.2010.21. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5463616.

[20] T. F. Van de Mortel, "Faking it: Social desirability response bias in self-report research," *Australian Journal of Advanced Nursing, The*, vol. 25, no. 4, pp. 40–48, 2008.

[21] E. Hassan, "Recall bias can be a threat to retrospective and prospective research designs," *The Internet Journal of Epidemiology*, vol. 3, no. 2, pp. 339–412, 2006.

[22] C. Ebert, M. Kuhrmann, and R. Prikladnicki, "Global software engineering: Evolution and trends," *IEEE 11th International Conference on Global Software Engineering*, 2016. DOI: 10.1109/ICGSE.2016.19. [Online]. Available: https://ieeexplore.ieee.org/document/7752721.

[23] A. Deshpande, H. Sharp, L. Barroca, and P. Gregory, "Remote working and collaboration in agile teams," 2016, Accessed: 2025-02-28. [Online]. Available: https://oro.open.ac.uk/47461/.

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

[25] E. Ó Conchúir, P. J. Ågerfalk, H. H. Olsson, and B. Fitzgerald, "Global software development: Where are the benefits?" *Communications of the ACM*, vol. 52, no. 8, pp. 127–131, 2009. DOI: 10.1145/1536616.1536648.

[26] C. Ebert, M. Kuhrmann, and R. Prikladnicki, "Global software engineering: Evolution and trends," *IEEE 11th International Conference on Global Software Engineering*, 2016. DOI: 10.1109/ICGSE.2016.19. [Online]. Available: https://ieeexplore.ieee.org/document/7752721.

[27] G. K. Stahl, M. L. Maznevski, A. Voigt, and K. Jonsen, "Unraveling the effects of cultural diversity in teams: A meta-analysis of research on multicultural work groups," *Journal of international business studies*, vol. 41, pp. 690–709, 2010.

[28] B. Hunley, S. Chakraborty, and S. MacDonald, "The impact of cultural communication on team performance," 2018.

[29] I. Zada, S. Shahzad, and S. Nazir, "Issues and implications of scrum on global software development," *Bahria University Journal of Information Communication Technologies*, vol. 8, no. 1, pp. 81–87, 2015, Accessed: 2025-02-28. [Online]. Available: https://www.researchgate.net/publication/273193605_Issues_and_implication_of_scrum_on_global_software_development.

[30] A. A. Khan, M. Shameem, M. Nadeem, and M. A. Akbar, "Agile trends in chinese global software development industry: Fuzzy ahp based conceptual mapping," *Applied Soft Computing Journal*, vol. 102, 2021, Accessed: 2025-02-28. DOI: 10.1016/j.asoc.2021.107090. [Online]. Available: https://doi.org/10.1016/j.asoc.2021.107090.

[31] N. Saleem, S. Mathrani, and N. Taskin, "Understanding the different levels of challenges in global software development," *IEEE Xplore*, 2019. DOI: 10.1109/ICGSE.2019.8807743.

[32] E. MacGregor, Y. Hsieh, and P. Kruchten, "The impact of intercultural factors on global software development," in *Canadian Conference on Electrical and Computer Engineering, 2005.*, IEEE, 2005, pp. 920–926.

[33] G. Hofstede, "Dimensionalizing cultures: The hofstede model in context," *Online Readings in Psychology and Culture*, vol. 2, no. 1, p. 8, Dec. 2011. DOI: 10.9707/2307-0919.1014. [Online]. Available: https://scholarworks.gvsu.edu/orpc/vol2/iss1/8.

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

[35] W. Mangundjaya, I. Gandakusuma, and M. S. Arumi, “Power distance, uncertainty avoidance and commitment to change,” *Jurnal Scientia*, vol. 12, no. 03, pp. 2852–2860, 2023.

[36] A. Unknown, “The role of intercultural differences and challenges faced in negotiating active mine sites’ rehabilitation objectives from africa to europe,” *Journal of Environmental Management*, pp. 2–4, 16, 2023, Accessed: 2025-02-28. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214790X2300151X.

[37] T. Likhi, “Effects of the power distance index on multicultural software engineering teams,” pp. 19–25, 2022, Discussion on unconscious biases in work distribution, communication, and cultural adaptation. [Online]. Available: https://theses.liacs.nl/pdf/2021-2022-LikhiT.pdf.

[38] S. Matthiesen and P. Bjørn, “Implicit bias and negative stereotyping in global software development and why it is time to move on!” *Journal of Software: Evolution and Process*, vol. 35, no. 2, e2435, 2023. DOI: 10.1002/smr.2435. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2435.

[39] D. Welsch, L. Burk, D. Mötefindt, and M. Neumann, “Navigating cultural diversity: Barriers and potentials in multicultural agile software development teams,” *arXiv preprint arXiv:2311.12061*, 2023.

[40] L. Mundere and P. Marešová, “Potential of ict industry for economic growth in developing countries - case study of rwanda,” *Advanced Science Letters*, pp. 48–50, 56, 2016. [Online]. Available: https://theses.cz/id/wccx2k/STAG84980.pdf.

[41] R. Sitas, L. Cirolia, A. Pollio, and A. G. Sebarenzi, “Platform politics and silicon savannahs: The rise of on-demand logistics and mobility in nairobi and kigali,” *International Centre for Cities*, 2022. [Online]. Available: 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-de

mand-logistics-and-mobility-in-Nairobi-and-Kigali.pdf.

[42] C. A. K. Gbali, "Impact of national policies on knowledge transfer to small tech companies: The case of rwanda," *GRIPS Discussion Paper*, pp. 4–5, 2021. [Online]. Available: https://grips.repo.nii.ac.jp/record/2000139/files/k295_report_phd20605.pdf.

[43] TechGyant. "Ict powers rwanda's gdp with 3.5% all-time high growth in 2023." Accessed: 13 May 2025, TechGyant. (2023), [Online]. Available: https://techgyant.com/ict-powers-rwandas-gdp-with-35-all-time-high-growth-in-2023/ (visited on 05/13/2023).

[44] Ministry of ICT and National Guidance, "Moict emagazine," Ministry of ICT and National Guidance, Uganda, Government Publication Issue 001, 2024, Accessed: 13 May 2023. [Online]. Available: https://ict.go.ug/site/documents/MoICT_eMag_Issue001_2024.pdf (visited on 05/13/2023).

[45] E. Bainomugisha, R. Hebig, and M. R. V. Chaudron, "Emerging software engineering research networks in (east) africa," *ACM SIGSOFT Software Engineering Notes*, vol. 46, pp. 18–22, 2021. DOI: 10.1145/3448992.3448996. [Online]. Available: https://consensus.app/papers/emerging-software-engineering-research-networks-in-east-bainomugisha-hebig/74100fba99b451bbac64bf5c7e2120a0/?utm_source=chatgpt.

[46] C. Otioma, "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*, Accessed: May 13, 2025, African Development Bank, Sharm El Sheikh, Egypt, Dec. 2019. [Online]. Available: 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.

[47] E. K. Mwangi, L. X. W. Thuku, and J. P. Kangethe, *Software development industry in east africa: Knowledge management perspective and*

value proposition, https://arxiv.org/abs/1504.02017, arXiv preprint arXiv:1504.02017, 2015.

[48] Norrsken Foundation, *Norrsken house kigali*, Accessed: 2025-01-27, 2025. [Online]. Available: https://www.norrsken.org/houses/kigali.

[49] M. Devlin and C. Phillips, "Assessing competency in undergraduate software engineering teams," in *IEEE EDUCON 2010 Conference*, IEEE, 2010, pp. 271–278.

[50] A. Sahakyan and T. Sloyan, "General coding skills evaluation framework: Technical research paper," CodeSignal, Tech. Rep., Apr. 2023, Originally published July 2019; Updated April 2023. [Online]. Available: https://discover.codesignal.com/rs/659-AFH-023/images/General-Coding-Framework-Technical-Research-Paper-CodeSignal.pdf.

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

[52] J. S. McLaughlin, G. W. McLaughlin, and J. A. Muffo, "Using qualitative and quantitative methods for complementary purposes: A case study," *New directions for institutional research*, vol. 2001, no. 112, pp. 15–44, 2001.

[53] P. Lenberg, R. Feldt, L. Gren, L. G. Wallgren Tengberg, I. Tidefors, and D. Graziotin, "Qualitative software engineering research: Reflections and guidelines," *Journal of Software: Evolution and Process*, vol. 36, no. 6, e2607, 2024.

[54] M. D. Penta and D. A. Tamburri, "Combining quantitative and qualitative studies in empirical software engineering research," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017, pp. 499–500. DOI: 10.1109/ICSE-C.2017.163. [Online]. Available: https://doi.org/10.1109/ICSE-C.2017.163.

[55] A. E. Betzner, "Pragmatic and dialectic mixed method approaches: An empirical comparison," Adviser: Frances Lawrenz, Ph.D. dissertation, University of Minnesota, Minneapolis, MN, Dec. 2008.

[56] N. Assyne, H. Ghanbari, and M. Pulkkinen, “The essential competencies of software professionals: A unified competence framework,” *Information and Software Technology*, vol. 151, pp. 14–15, 2022. DOI: 10.1016/j.infsof.2022.107020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584922001446.

[57] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, and L. Li, “Large language models for software engineering: A systematic literature review,” *ACM Transactions on Software Engineering*, 2024. [Online]. Available: https://arxiv.org/pdf/2308.10620.

[58] A. Moharil and A. Sharma, “Tabasco: A transformer-based contextualization toolkit,” *Science of Computer Programming*, vol. 230, p. 102 994, 2023. DOI: 10.1016/j.scico.2023.102994. [Online]. Available: https://doi.org/10.1016/j.scico.2023.102994.

[59] Y. Dyulicheva and E. Bilashova, “Learning analytics of moocs based on natural language processing,” *CS&SE@SW*, pp. 2–3, 5–7, 10–13, 2021. [Online]. Available: https://www.researchgate.net/profile/Yulia-Dyulicheva/publication/357173866_Learning_Analytics_of_MOOCs_based_on_Natural_Language_Processing/links/61e8305e8d338833e37dff0e/Learning-Analytics-of-MOOCs-based-on-Natural-Language-Processing.pdf.

[60] A. Huang and S. Yang, “Dna of learning behaviors: A novel approach of learning performance prediction by nlp,” *Artificial Intelligence*, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666920X24000286.

[61] Z. Feng *et al.*, “Codebert: A pre-trained model for programming and natural languages,” *arXiv preprint arXiv:2002.08155*, 2020.

[62] G. Jawahar, B. Sagot, and D. Seddah, “What does bert learn about the structure of language?” *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[63] J. Martinez-Gil, “Augmenting the interpretability of GraphCodeBERT for code similarity tasks,” *arXiv preprint arXiv:2410.05275*, 2024. arXiv: 2410.05275 [cs.IR].

[64] M. Obaidi and J. Klünder, “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.

[65] M. Ilyas, T. Qadah, and A. Ghaffar, “Factors influencing software development competencies of gsd teams: Extraction from slr and empirical study,” *IEEE Access*, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10804802/.

[66] A. P. Oluleye and A. A. Mayowa, “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*, vol. 5, no. 3, pp. 41–57, 2025.

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

[68] S. Sahay, B. Nicholson, and S. Krishna, *Global IT outsourcing: software development across borders*. Cambridge University Press, 2003.

[69] A. Alsahli, H. Khan, and S. Alyahya, “Agile development overcomes gsd challenges: A systematic literature review,” *International Journal of Computer Science and Software Engineering*, vol. 6, no. 1, p. 7, 2017.

[70] P. Hanafizadeh and A. Zareravasan, “A systematic literature review on it outsourcing decision and future research directions,” *Journal of Global Information Management (JGIM)*, vol. 28, no. 2, pp. 160–201, 2020.

[71] M. Lukauskas and V. Pilinkienė, “Enhancing skills demand understanding through job ad segmentation using nlp and clustering techniques,” *Applied Sciences*, pp. 2–3, 5, 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/10/6119.

[72] C. Hiranrat and A. Harncharnchai, "Using text mining to discover skills demanded in software development jobs in thailand," *Proceedings of the 2nd international conference on data science and information technology*, pp. 33–37, 2018. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3206129.3239426.

[73] T. Lehtonen, T. Aho, K. Kuusinen, and T. Mikkonen, "Visualizations for software development process management," in *Information Modelling and Knowledge Bases XXVIII*, IOS Press, 2017, pp. 1–12.

[74] S. Diehl, "Software visualization," in *Proceedings of the 27th international conference on Software engineering*, 2005, pp. 718–719.

[75] CoderPad. "Coderpad's tech hiring survey finds 60% of recruiters are ready to ditch the cv, 40% looking internationally to find the best candidates in tough job market." Accessed: May 15, 2025, BusinessWire. (Jan. 2022), [Online]. Available: https://www.businesswire.com/news/home/20220109005015/en/CoderPads-Tech-Hiring-Survey-Finds-60-of-Recruiters-Are-Ready-to-Ditch-the-CV-40-Looking-Internationally-to-Find-the-Best-Candidates-in-Tough-Job-Market.

[76] CodinGame and CoderPad. "Codingame and coderpad tech hiring survey 2023." Accessed: May 15, 2025, CodinGame. (2023), [Online]. Available: https://www.codingame.com/work/codingame-and-coderpad-tech-hiring-survey-2023/.

[77] HackerRank. "Companies using hackerrank for technical hiring." Accessed: May 15, 2025, HackerRank. (2025), [Online]. Available: https://www.hackerrank.com/customers/.

[78] IEEE Computer Society, "Software engineering competency model (swecom)," IEEE Computer Society, Technical Standard, version 1.0, 2014, A comprehensive framework defining competency areas for software engineering professionals across multiple levels of expertise. [Online]. Available: https://www.computer.org/volunteering/boards-and-committees/professional-educational-activities/software-engineering-competency-model.

[79] R. Florea, “The software tester: An exploration of the skills and practice of the role,” *University of Oslo*, 2023. [Online]. Available: https://www.duo.uio.no/bitstream/handle/10852/101865/1/PhD-Florea-2023.pdf.

[80] M.-A. Storey, R. Hoda, A. M. P. Milani, and M. T. Baldassarre, “Guiding principles for mixed methods research in software engineering,” *arXiv preprint arXiv:2404.06011*, pp. 4, 6–7, 2024.

[81] E. Clarke and J. Visser, “Pragmatic research methodology in education: Possibilities and pitfalls,” *International Journal of Research & Method in Education*, vol. 42, no. 5, pp. 455–469, 2019.

[82] G. Edwards, *Education and theory: Strangers in paradigms*, 2007.

[83] M. Karanja, L. X. Thuku, and J. P. Kangethe, “Software development industry in east africa: Knowledge management perspective and value proposition,” *Africa Casebook - Synergies in African Business and Management Practices*, pp. 26–37, 2015. [Online]. Available: https://www.researchgate.net/publication/274730121_Software_Development_Industry_In_East_Africa_Knowledge_Management_Perspective_And_Value_Proposition.

[84] M. Hennink and B. N. Kaiser, “Sample sizes for saturation in qualitative research: A systematic review of empirical tests,” *Social Science & Medicine*, vol. 292, pp. 2, 6, 2022. DOI: 10.1016/j.socscimed.2021.114523.

[85] E. Dagan, A. Sarma, A. Chang, S. D’Angelo, J. Dicker, and E. Murphy-Hill, “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*, 2023, pp. 631–643.

[86] G. Guest, A. Bunce, and L. Johnson, “How many interviews are enough? an experiment with data saturation and variability,” *Field methods*, vol. 18, no. 1, pp. 59–82, 2006.

[87] P. Sedgwick, “Convenience sampling,” *Bmj*, vol. 347, 2013.

[88] W. G. Cochran, “Sampling techniques,” *Johan Wiley & Sons Inc*, 1977.

[89] Z-Table.com. “90% confidence interval z-score.” Statistical reference for confidence interval calculations, Z-Table.com. (2025), [Online]. Available: https://z-table.com/90-confidence-interval-z-score.html (visited on 05/15/2025).

[90] A. Karasz and T. M. Singelis, *Qualitative and mixed methods research in cross-cultural psychology*, 2009.

[91] J. W. Creswell, A. C. Klassen, V. L. Plano Clark, K. C. Smith, *et al.*, “Best practices for mixed methods research in the health sciences,” *Bethesda (Maryland): National Institutes of Health*, vol. 2013, pp. 541–545, 2011.

[92] L. Hespanhol, C. S. Vallio, L. M. Costa, and B. T. Saragiotto, “Understanding and interpreting confidence and credible intervals around effect estimates,” *Brazilian journal of physical therapy*, vol. 23, no. 4, pp. 290–301, 2019.

[93] B. Giraudeau, A. Caille, S. M. Eldridge, C. Weijer, M. Zwarenstein, and M. Taljaard, “Heterogeneity in pragmatic randomised trials: Sources and management,” *BMC medicine*, vol. 20, no. 1, p. 372, 2022.

[94] R. Colomo-Palacios, E. Tovar-Caro, and Á. García-Crespo, “Identifying technical competences of it professionals: The case of software engineers,” Universidad Carlos III de Madrid / Universidad Politécnica de Madrid, Tech. Rep., 2018, Also known in Spanish as “Niveles competenciales para los perfiles de Ingenieros de Software”.

[95] C. Zhou, S. K. Kuttal, and I. Ahmed, “What makes a good developer? an empirical study of developers’ technical and social competencies,” in *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, IEEE, 2018, pp. 319–321.

[96] LeetCode, *Two Sum*, https://leetcode.com/problems/two-sum/, Accessed: 2025-05-05.

[97] S. Jindal, “Why faang companies ask leetcode,” *Medium*, Dec. 2024, Accessed: 2025-05-05. [Online]. Available: https://medium.com/write-a-catalyst/why-leetcode-is-so-popular-82bb94c97827.

[98] Z. Wan, Y. Zhang, X. Xia, Y. Jiang, and D. Lo, “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*, 2023, pp. 1457–1469.

[99] A. Bacchelli and C. Bird, “Expectations, outcomes, and challenges of modern code review,” in *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, Foundational study on modern code review practices, covering goals like quality improvement, finding defects, knowledge transfer, team awareness, and associated challenges., 2013, pp. 1–2. DOI: 10.1109/ICSE.2013.6606647.

[100] H. Assal, S. G. Morkonda, M. Z. Arif, and S. Chxiasson, “Software security in practice: Knowledge and motivation,” *Journal of Cybersecurity*, 2025. DOI: 10.1093/cybsec/tyaf005. [Online]. Available: https://doi.org/10.1093/cybsec/tyaf005.

[101] A. N. Meyer, E. T. Barr, C. Bird, and T. Zimmermann, “Today was a good day: The daily life of software developers,” *IEEE Transactions on Software Engineering*, vol. 47, no. 5, pp. 863–880, 2019.

[102] S. A. Licorish and S. G. MacDonell, “Exploring software developers’ work practices: Task differences, participation, engagement, and speed of task resolution,” *Information & Management*, vol. 54, no. 3, pp. 364–382, 2017.

[103] M. Chen and A. Golan, “What may visualization processes optimize?” *IEEE transactions on visualization and computer graphics*, vol. 22, no. 12, pp. 2619–2632, 2015.

[104] M. K. Kerr and G. A. Churchill, “Bootstrapping cluster analysis: Assessing the reliability of conclusions from microarray experiments,” *Proceedings of the national academy of sciences*, vol. 98, no. 16, pp. 8961–8965, 2001.

[105] S. Monti, P. Tamayo, J. Mesirov, and T. Golub, “Consensus clustering: A resampling-based method for class discovery and visualization of gene expression microarray data,” *Machine learning*, vol. 52, pp. 91–118, 2003.

[106] G. Ogbuabor and F. Ugwoke, “Clustering algorithm for a healthcare dataset using silhouette score value,” *Int. J. Comput. Sci. Inf. Technol*, vol. 10, no. 2, pp. 27–37, 2018.

[107] K. R. Shahapure and C. Nicholas, “Cluster quality analysis using silhouette score,” in *2020 IEEE 7th international conference on data science and advanced analytics (DSAA)*, IEEE, 2020, pp. 747–748.

[108] S. Lubbe, “Bootstrapping cluster analysis solutions with the r package clusboot,” *Austrian Journal of Statistics*, vol. 53, no. 3, pp. 1–19, 2024.

[109] S. E. Stemler, “A comparison of consensus, consistency, and measurement approaches to estimating interrater reliability,” *Practical Assessment, Research, and Evaluation*, vol. 9, no. 1, 2004.

[110] J. Cohen, *Statistical power analysis for the behavioral sciences*. routledge, 2013.

[111] C. R. Brydges, “Effect size guidelines, sample size calculations, and statistical power in gerontology,” *Innovation in aging*, vol. 3, no. 4, igz036, 2019.

[112] J. W. Creswell and V. L. P. Clark, *Designing and conducting mixed methods research*. Sage publications, 2017.

[113] M. Kitani and H. Murakami, “One-sample location test based on the sign and wilcoxon signed-rank tests,” *Journal of Statistical Computation and Simulation*, vol. 92, no. 3, pp. 610–622, 2022.

[114] S. Baltes and P. Ralph, “Sampling in software engineering research: A critical review and guidelines,” *Empirical Software Engineering*, vol. 27, no. 4, p. 94, 2022.

[115] C. Ebert and P. De Neve, “Surviving global software development,” *IEEE software*, vol. 18, no. 2, pp. 62–69, 2001.

[116] Times Higher Education. “World university rankings 2025.” Filtered by: Netherlands, Rwanda, Sweden, Uganda; Subject: 3081, Times Higher Education. (2025), [Online]. Available: https://www.timeshighereducation.com/world-university-rankings/latest/world-ranking#!/length/25/locations/NLD+RWA+SWE+UGA/subjects/3081/sort_by/rank/sort_order/asc/cols/scores (visited on 05/07/2025).

[117] BMS Performance. “The importance of hiring for potential, not experience.” (Aug. 2024), [Online]. Available: https://bmsperformance.com/insight/the-importance-of-hiring-for-potential-not-experience/ (visited on 05/07/2025).

[118] N. Basabe and M. Ros, “Cultural dimensions and social behavior correlates: Individualism-collectivism and power distance,” *International Review of Social Psychology*, vol. 18, no. 1, pp. 189–225, 2005.

TRITA – XXX-EX 2025:0000
Stockholm, Sweden 2025

www.kth.se

€€€€ For DIVA €€€€

{
”Author1”: { ”Last name”: ”Khosravi”,
”First name”: ”Sam”,
”Local User Id”: ”u1tukeft”,
”E-mail”: ”smkh@kth.se”,
”organisation”: {”L1”: ””,
}
},
”Cycle”: ”2”,
”Course code”: ”DA231X”,
”Credits”: ”30.0”,
”Degree1”: {”Educational program”: ”Master’s Programme, Computer Science, 120 credits”
,”programcode”: ”TCSCM”
,”Degree”: ”Degree of Master of Science in Engineering”
,”subjectArea”: ”Computer Science and Engineering”
},
”Title”: {
”Main title”: ”A Comparative Study of Software Engineers in East Africa and North Western Europe Based on Skills, Communication, Organizational Culture, and Perceptions”,
”Subtitle”: ”A Pragmatic Mixed-Methods Approach to Understanding Global Software Development Dynamics Between Rwanda, Uganda, Sweden, and the Netherlands”,
”Language”: ”eng” },
”Alternative title”: {
”Main title”: ”En jämförande studie av mjukvaruingenjörer i Östafrika och Nordvästeuropa baserat på färdigheter, kommunikation, organisationskultur och uppfattningar”,
”Subtitle”: ”En pragmatisk mixed-methods strategi för att förstå global mjukvaruutveckling mellan Rwanda, Uganda, Sverige och Nederländerna”,
”Language”: ”swe”
},
”Supervisor1”: { ”Last name”: ”Payerah”,
”First name”: ”Amir H.”,
”Local User Id”: ”u1a73o9d”,
”E-mail”: ”payberah@kth.se”,
”organisation”: {”L1”: ””,
”L2”: ”EECS” }
},
”Supervisor2”: { ”Last name”: ”Bukhsh”,
”First name”: ”Faiza A.”,
”E-mail”: ”f.a.bukhsh@utwente.nl”,
”Other organisation”: ”University of Twente, Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)”
},
”Examiner1”: { ”Last name”: ”Engwall”,
”First name”: ”Olov”,
”Local User Id”: ”u1niocbs”,
”E-mail”: ”engwall@kth.se”,
”organisation”: {”L1”: ””,
”L2”: ”EECS” }
},
"National Subject Categories": "10201, 10206",
"SDGs": "8, 9",
”Other information”: {”Year”: ”2025”, ”Number of pages”: ”xiii,87”},
”Copyrightleft”: ”copyright”,
”Series”: { ”Title of series”: ”TRITA – XXX-EX” , ”No. in series”: ”2025:0000” },
”Opponents”: { ”Name”: ”XXX”},
”Presentation”: { ”Date”: ”2022-03-15 13:00”
,”Language”:”XXX”
,”Room”: ”via Zoom https://kth-se.zoom.us/j/ddddddddddd”
,”Address”: ”Isafjordsgatan 22 (Kistagången 16)”
,”City”: ”Stockholm” },
”Number of lang instances”: ”2”,
”Abstract[eng]”: €€€€

This research examines cross-cultural dynamics in global software development by comparing software engineers from East Africa (Rwanda and Uganda) and North Western Europe (Sweden and the Netherlands). Despite growing African tech hubs, empirical research looking into the technical competencies, communication styles, and organizational practices of East African software engineers remain limited, despite potential time-zone alignment advantages with European partners. Most existing studies on global software development focus on established outsourcing destinations in Asia, Eastern Europe, and South America.

This thesis addresses this research gap by using a pragmatic mixed-methods approach, combining quantitative and qualitative data, which was collected through standardized programming challenges, system design tasks, code reviews, and in-depth interviews with 48 software engineers across the four countries. Technical solutions were analyzed using Machine Learning techniques, including CodeBERT embeddings and clustering analysis, while communication styles were evaluated through sentiment analysis of code reviews using Valence Aware Dictionary and Sentiment Reasoner from the Natural Language Toolkit.

The results reveal insights challenging conventional assumptions about global software engineering. Rwandan senior software engineers showed technical skills comparable to their European counterparts, while Rwandan junior software engineers were outclassed by their European counterparts. Communication styles differed across regions, with East African software engineers providing more positive and supportive feedback (compound sentiment scores >0.5) compared to the more critical, direct approach of European software engineers (compound score −0.361 for Dutch software engineers). Process priorities also varied as European teams allocated more time to initial development and design, while East African teams put more focus on code reviews and deployment processes.
Most significantly, the study found perception gaps with European software engineers consistently underrating the capabilities of East Africans despite measured performance showing individual excellence and overlap between regions in terms of skills. These findings provide empirical evidence that can inform more equitable global software development practices and enhance cross-cultural collaboration by leveraging the complementary strengths adherent in each region. Furthermore it challenges biases that may limit opportunities for talented software engineers from emerging regions, and contributes to a more inclusive understanding of global software engineering dynamics in previously understudied contexts.

”Abstract[swe]”: €€€€

Denna studie undersöker tvärkulturell dynamik inom global mjukvaruutveckling genom att jämföra mjukvaruingenjörer från Östafrika mot de från Nordvästa Europa. I studien ingår Rwanda och Uganda som skall representera Östafrika, medan Sverige and Nederländerna representerar Nordvästra Europa. Trots att Östafrikanska länder utvecklat både infrastruktur och teknisk kompetens de senaste åren saknas det empirisk forskning om östafrikanska utvecklares tekniska kompetenser, kommunikationssätt och organisatoriska metoder, särskilt i jämförelse med andra mer etablerade outsourcing destinationer i Asien och Östeuropa.

Studien omfattar 48 utvecklare, och använder en pragmatisk mixed-methods ansats som kombinerar kvantitativ och kvalitativ data. Datainsamlingen omfattade standardiserade programmeringsutmaningar, systemdesign, kodgranskningar och djupintervjuer. Maskininlärningsmetoder såsom CodeBERT-embeddings och klusteranalys användes för att analysera tekniska lösningar, medan kommunikationsstilar utvärderades genom sentimentanalys med VADER från NLTK.

Resultaten visar att seniora utvecklare från Rwanda besitter tekniska färdigheter som är jämförbara med deras europeiska motsvarigheter, däremot presterar juniora utvecklare från Rwanda sämre. Kommunikationsstilar skiljer sig markant mellan de två regionerna, då östafrikanska utvecklare gav mer positiv feedback på kodgranskningar, jämfört med européers mer kritiska och direkta tillvägagångssätt. Processprioriteringar varierade också, där europeiska team fokuserade mer på initial utveckling medan östafrikanska team fokuserade mer på kodgranskning och driftsättning.
Mest anmärkningsvärt var de perceptions skillnader som hittades, där europeiska utvecklare konsekvent underskattade östafrikanska utvecklares förmågor trots uppmätta prestationer som visade individuell förmåga och överlapp mellan regionerna. Dessa fynd ger empiriska bevis som kan bidra till mer rättvisa globala mjukvaruutvecklings möjligheter och utmana fördomar som begränsar möjligheter för talangfulla utvecklare från tillväxtregioner.

€€€€,
”Keywords[eng]”: €€€€
Global Software Development, Cross-cultural Collaboration, Software Engineering Skills, East Africa, North Western Europe, Perception Bias €€€€,
€€€€,
”Keywords[swe]”: €€€€
Global Mjukvaruutveckling, Tvärkulturellt Samarbete, Mjukvaruingenjörsfärdigheter, Östafrika, Nordvästeuropa, Uppfattningsbias €€€€,
}

acronyms.tex

```
%%% Local Variables:
%%% mode: latex
%%% TeX-master: t
%%% End:
% The following command is used with glossaries-extra
\setabbreviationstyle[acronym]{long-short}
% The form of the entries in this file is \newacronym{label}{acronym}{phrase}
%                                      or \newacronym[options]{label}{acronym}{phrase}
% see "User Manual for glossaries.sty" for the  details about the options, one example is shown below
% note the specification of the long form plural in the line below
%%% Local Variables:
%%% mode: latex
%%% TeX-master: t
%%% End:
% note the use of a non-breaking dash in long text for the following acronym
% Define acronyms in alphabetical order
\newacronym{2FA}{2FA}{Two-Factor Authentication}
\newacronym{API}{API}{Application Programming Interface}
\newacronym{BERT}{BERT}{Bidirectional Encoder Representations from Transformers}
\newacronym{CI}{CI}{Confidence Interval}
\newacronym{CI/CD}{CI/CD}{Continuous Integration/Continuous Deployment}
\newacronym{codeBERT}{codeBERT}{Code Bidirectional Encoder Representations from Transformers}

\newacronym{DevOps}{DevOps}{Development and Operations}
\newacronym{EA}{EA}{East Africa}
\newacronym{GSD}{GSD}{Global Software Development}
\newacronym{ICT}{ICT}{Information and Communications Technology}
\newacronym{ML}{ML}{Machine Learning}
\newacronym{NLP}{NLP}{Natural Language Processing}
\newacronym{NLTK}{NLTK}{Natural Language Toolkit}
\newacronym{NWE}{NWE}{North Western Europe}
\newacronym{PCA}{PCA}{Principal Component Analysis}
\newacronym{RQ}{RQ}{Research Question}
\newacronym{SDG}{SDG}{Sustainable Development Goals}
\newacronym{SWEBOK}{SWEBOK}{Software Engineering Body of Knowledge}
\newacronym{VADER}{VADER}{Valence Aware Dictionary and Sentiment Reasoner}
```

All SDGs are within the range [1-17].