ORIGINAL ARTICLE

The dynamics of strengthening the skills to develop safety-critical systems in Tanzania

Leonard Peter Binamungu[a], Salome Maro[a] Godfrey Justo[a] and Jasson Ndanguzi[b]

[a]Department of Computer Science and Engineering, College of Information and Communication Technologies, University of Dar es Salaam, Dar es Salaam, Tanzania; [b]Tanzania Information and Communication Technologies Commission

ARTICLE HISTORY
Compiled May 1, 2025

ABSTRACT
Background and Context: Safety-critical systems are systems whose failure could cause injury, death, or damage to property or the environment. Examples include health systems, aircraft flight control systems, weapons, and nuclear systems. The development of software for safety-critical systems demands safety considerations throughout the software development lifecycle, especially during design and implementation. As such systems become prevalent around the world, including in developing countries, the skills to develop them become indispensable. However, such skills are scarce in the developing world, due to the limitations of university ICT curricula. Existing studies on software safety training for software professionals are not sensitive to the context of the developing world and do not cover the design and implementation of the curriculum.
Objective: To establish the skills required to develop safety-critical systems in the context of developing countries and the suitable strategies for teaching different topics on software safety in the context of developing countries.
Method: We surveyed 70 software practitioners and conducted 25 follow-up interviews to establish software safety skill gaps in Tanzania, and used the established skill gaps to develop a context-sensitive software safety curriculum for software practitioners in Tanzania. We also trained 80 software practitioners on software safety across four training workshops, gathering in-training feedback.
Findings: A context-sensitive software safety curriculum, whose implementation enabled us to learn, among other things, that context-sensitive curriculum content, examples and adult learning techniques could improve learning outcomes on software safety in Tanzania and, by extension, similar developing countries.
Implications: The findings underscore the importance of context-sensitive software safety content and pedagogy for developing countries. They also provide opportunities to theorize and benchmark the skills required to develop safety-critical systems in developing countries.

KEYWORDS
software safety; safety-critical systems; contextualization; software engineering curriculum; software professionals training; developing countries

1. Introduction

The safety of engineering products is an important aspect because failure of safety-critical engineering products can lead to accidents, injuries, deaths, and significant

CONTACT Leonard Peter Binamungu Email: lepebina@udsm.ac.tz

damage to property or the environment. Producing safe engineering products demands consideration of safety throughout the product engineering process, particularly during requirements analysis, design and implementation. As systems that consist of software-controlled hardware are increasingly used in different domains such as health and automotive, software safety becomes a crucial aspect, because software failures can lead to system safety violations, which could potentially cause accidents, injuries, deaths, or significant damage to property and the environment. However, due to little or no emphasis on software safety knowledge and skills in university ICT (Information and Communication Technologies) curricula in most developing countries, software engineers in the developing world lack the requisite knowledge and skills to develop safety-critical systems. For example, currently, there is little or no safety coverage in the computer science/engineering curricula of the universities and other higher learning institutions in Tanzania, leading to the production of software engineers who lack knowledge and skills on how to consider safety aspects when designing and implementing software systems.

The work of Ruwodo et al. (2022) suggested the revamping of the software engineering education in African higher education because of the high demand for skilled professionals, the dictates of the Fourth Industrial Revolution (4IR), and the extant situation where universities produce software engineers that cannot serve in the job markets. Indeed, in the 4IR, fundamental shifts are taking place in how the global production and supply network operates through ongoing automation of traditional manufacturing and industrial practices, using modern smart technology, large-scale machine-to-machine communication, and the Internet of Things. This integration results in increasing automation, improving communication and self-monitoring, and the use of smart machines that can analyse and diagnose issues without the need for human intervention (Moore, 2019). In some application contexts, the failure of a system can have adverse effects on users, thus making it important for software engineers to foresee potential failure scenarios and plan for appropriate mitigation strategies to avert harm. As such computers play a significant role in operating many modern systems, some of which are classified as safety-critical, any failure in safety-critical systems may result in injuries, loss of life, or significant damage to property or the environment. Examples include medical systems, aircraft flight control systems, weapons and nuclear systems. When designing such systems, which usually include both software and hardware, the most important factor is safety (Rafeh and Rabiee, 2013).

Moreover, existing efforts to train software professionals in the development of safety-critical systems (Kang and Do, 2021; Stroud and Pfitzer, 2016) are mostly informed by the context of the developed world, when safety-critical systems that are mostly developed by software engineers in the developing world could differ significantly from safety-critical systems developed by software engineers in the developed world. For example, given the less-industrialized nature of most developing countries, software engineers in most developing countries hardly get involved in the development of software for controlling safety-critical systems such as aircraft and vehicles. However, they are most likely to be involved in the development of safety-critical systems in the health domain whose failure could also have catastrophic effects. For example, in Tanzania, most hospital information management systems, laboratory information management systems, and software that interface with medical devices for automatic administration of drugs to patients are developed locally, either by small software development companies or by the government using local developers. Another example in the healthcare domain is the existence of information systems integrated with medical devices, such as software designed to calculate radiation doses for cancer patients.

This calls for the need to contextualize software safety training, as is required for the rest of the software engineering education (Fendler and Winschiers-Theophilus, 2010; Osman and Dias-Neto, 2014; Osman, 2012). Existing studies on software safety training (Kang and Do, 2021; Stroud and Pfitzer, 2016) have only proposed the topics to be included in a software safety training program. They, however, have not covered the design, execution, and evaluation of a curriculum on software safety, lacking empirical lessons on the actualization of the recommended topics on software safety.

Importantly, the process of deciding what to include in the software safety curriculum and how to deliver such a curriculum for software professionals in a developing country is non-trivial. It involves different context-sensitive dynamics that must be carefully considered for a successful software safety training program. To the best of our knowledge, the existing literature has not covered such dynamics. To contribute to closing this gap, the present paper reflects our experiences of designing and implementing a software safety curriculum for software practitioners in Tanzania, generating lessons that could apply to other developing countries. Skills gap analysis was conducted to understand the specific software safety knowledge and skills required by software engineering practitioners in Tanzania. The skills gap analysis produced the results that informed the design of a curriculum that addresses the knowledge/skills needed for the design and implementation of safety-critical software, targeting early career software engineering graduates and practitioners. As a proof of concept, the designed curriculum was used in four training workshops of selected software practitioners in Tanzania, covering a total of 80 individuals, and therefore the study also draws the in-training lessons.

In particular, we seek to answer two research questions:

- **RQ1:** What skills for developing safety critical systems are required in the context of developing countries?
- **RQ2:** What teaching strategies are most suited for teaching different topics on software safety in the context of developing countries?

This paper builds on our previous work reported in Binamungu et al. (2024), which focused on identifying knowledge and skill gaps in software safety among software practitioners in Tanzania. The present paper makes an empirical contribution by providing a detailed account of designing and implementing a software safety curriculum for professional software engineers in Tanzania and, by extension, other similar developing countries. This empirical contribution implies opportunities to theorize and benchmark the skills required to develop safety-critical systems in developing countries.

The rest of the paper is structured as follows. Section 2 presents the literature related to this work, identifying the gap closed by the present study; Section 3 describes the methodology for the present study; Section 4 presents the results of the study, which are then discussed in Section 5. The implications of our empirical contribution for research and practice are also provided in Section 5. Section 6 concludes the paper.

2. Related Literature

In this section, we review the literature from two strands of work: the development of curricula for professional courses on software development, and experiences of teaching professional courses on systems development. The contributions of various studies in these areas are presented before highlighting the gap closed by the present paper.

2.1. Development of curricula for professional courses on software development

Literature in curriculum development shows that the curriculum development process usually has five main phases: needs analysis, development, validation, deployment, and monitoring and evaluation. Several studies have reported the methodologies for and experiences of curriculum development either focusing on software engineering generally (Zeid, 2007) or on specific topics of software engineering such as security (Ardis and Mead, 2011). For instance, the work of Kobata et al. (2014) proposed a curriculum development methodology which uses goal-oriented analysis. The methodology starts by defining strategic goals from industry and mapping these to educational goals and finally teaching methods. Their evaluation shows that the methodology leads to a curriculum that aligns with industry needs.

Our review of the existing literature focusing on the development of curricula for professional training on software safety reveals a notable gap: while techniques for functional safety and software safety have been heavily studied and documented in scientific literature, only a few studies report on steps taken in the curriculum design and design decisions made given a specific context. Closely related to our work is the work of Kang and Do (2021), which conducted a needs analysis for software safety training in South Korea and proposed high-priority topics for the software safety education program. Their analysis was done by surveying 259 software professionals. Another closely related work is the work of Stroud and Pfitzer (2016) that proposed the development of professional programs for safety and mission assurance professionals. However, their approach is that the professionals will decide what they want to learn and tailor the curriculum to their needs. The paper emphasizes training the professionals rather than focusing purely on course content. Given that many studies such as Fendler and Winschiers-Theophilus (2010), Zeid (2007) and Case et al. (2016) emphasize the need for contextualization even at the curriculum design level, we fill this gap by developing a contextualised software safety curriculum for a Tanzanian software engineering professional.

2.2. Experiences of teaching professional courses on software engineering

The studies reporting the experiences of teaching professional courses in software engineering can be put into three categories. The first category is made of papers that emphasize the importance of context in software engineering education (Fendler and Winschiers-Theophilus, 2010; Osman and Dias-Neto, 2014; Osman, 2012). The work of Fendler and Winschiers-Theophilus (2010), for example, used the experiences of teaching software engineering in Namibia to argue for the importance of context-sensitivity in software engineering education, emphasizing the differences between developed and developing countries and how such differences should also be reflected in software engineering education. They further proposed a framework for facilitating context-sensitive software engineering education. In a related study, Osman and Dias-Neto (2014) used a case of teaching a software engineering course to a mix of university students and industry professionals to amplify the importance of context-sensitive examples in teaching software engineering in Brazil. In a controlled experiment, real-life examples from the Brazillian context were found to facilitate more understanding among trainees compared to examples that were drawn from international textbooks on software engineering education. Software engineers trained through context-aware education systems also stand a good chance of building context-aware software systems that are advocated for in the work of Winschiers-Theophilus (2009). The studies reported by Blake and Tucker (2006); Blake (2010); Osman (2012) also emphasize the importance of training software engineers in such a way that they are always aware of the social context surrounding the systems they develop.

The second category of papers in this area focuses on techniques for teaching and evaluating the teaching of professional software engineers. The work of Ellis and Hislop (2004), for example, describes different techniques that have been found to work when training professional software engineers. These include entertaining self-directed learning, in which an instructor plays more of a facilitation role, instead of a leadership role; accommodating more practicals through problem-based learning, instead of focusing more on course content; using goal-based learning through modularization of the materials to be learned; and providing timely feedback, either through direct responses from the instructor(s) or indirect feedback through such arrangements as group discussions, in which students can contribute to a particular topic, enabling learners to share and learn from each other and instructors to gauge the level of understanding of the subject matter by students. The work of Striuk and Semerikov (2022) recommends lectures, practicals, projects, and independent studies as among the appropriate techniques for teaching software design to software engineers. They further recommend individual and group software design projects, in which the competencies developed by learners should be evaluated through the defence of the proposed solutions. Active learning techniques such as question-based learning, role-playing games and peer learning have also been recommended as appropriate for teaching software design concepts (Warren, 2005). Galster et al. (2018) recommended active watching of videos as another appropriate technique for developing appropriate competencies among professional software engineers. However, despite the importance of all these specific training techniques, Thomas et al. (2006) stress that the training needs and context should determine the techniques to be employed. This conclusion was informed by their eight years of experience in training working software professionals. The work of Kawano et al. (2019) used learner experiences to evaluate the effectiveness of a training program for professional software engineers, providing a technique for evaluating learning outcomes when training professional software engineers.

The third category of papers in this area has focused on software safety education for software engineers. In particular, there has been a general tendency to recognize the importance of providing specialised training for software professionals involved in the development of safety-critical systems (Kang and Do, 2021; Stroud and Pfitzer, 2016). For example, after proposing what to be included in a software safety curriculum, Kang and Do (2021) recommended conducting a post-training evaluation to inform the improvement of software safety training outcomes and the associated training program. Their study, however, did not cover the design, execution, and evaluation of a curriculum on software safety, lacking empirical lessons on the actualization of the recommended topics on software safety.

However, it has been observed in the reviewed literature that, while there is a reasonably adequate amount of coverage for the training needs of software engineering students in universities and institutions of higher learning, there is a paucity of empirical lessons drawn from teaching software engineering practitioners. Moreover, the training and evaluation techniques have also focused on general software engineering topics, leaving a knowledge gap on whether these techniques also work perfectly when training professional software practitioners on the development of safety-critical systems. Importantly, existing studies on software safety training focused on the context of the developed world. While safety engineers in the developed world might be involved in the development of software that control safety-critical systems such as aircraft, motor vehicles, nuclear reactors, self-driving cars, personal care robots, and several other examples that are common in the existing software safety courses, most developing countries are less industrialised, implying that software engineers in developing countries have negligible chances of developing software for controlling aircraft, vehicles, nuclear reactors, self-driving cars, personal care robots, etc. Instead, the developing world has its own common version of safety-critical systems, which should be emphasized when training software engineers from these countries on how to develop safety-critical systems. Our study shows that web-based information management systems are prevailing compared to embedded systems. A few embedded systems also exist which are developed using low cost sensors. Examples of such systems are systems to monitor environmental variables (such as temperature and humidity) in manufacturing plants. Thus, the existing literature not only lacks documented experiences and lessons in teaching software safety to software practitioners across the world, but it also lacks experiences and lessons in teaching software safety to software practitioners from the developing world. To contribute to closing this gap, we reflect on our experiences of teaching software safety to software practitioners in Tanzania, generating lessons that could apply to other developing countries.

3. Research methodology

A mixed-method approach was used for this study. Both quantitative and qualitative data were gathered, analysed and used for skills gap analysis, curriculum design, training of software practitioners on software safety, and evaluation of the training of software practitioners. In each case, qualitative data were used to complement quantitative data or facilitate a better interpretation of quantitative data.

3.1. Research design

We adopted an end-to-end processes cycle similar to an improvement cycle based on the scientific method that proposes a change in a process, implements a change, measures the results, and takes appropriate action (PDCA): this process is also known as the Deming Cycle (Schönsleben, 2023). We adapted the Deming Cycle to suit our context of designing and operationalizing a curriculum for a professional course on software safety. The 'Plan' part involved conducting the skills gap analysis using the survey and semi-structured interviews, establishing the curriculum for addressing the identified knowledge/skills needs and verifying the proposed curriculum through the stakeholders' workshop. The 'Do' part involved the delivery of the curriculum to selected software practitioners by conducting four training workshops on software safety. The 'Check/study' part involved gathering feedback from trainees at the end of each day for each of the four training workshops. The 'Act' part used a feedback loop mechanism that enabled the improvement of delivery in a subsequent training workshop. Figure 1 depicts the process activities of the study, as adapted from the PDCA cycle.

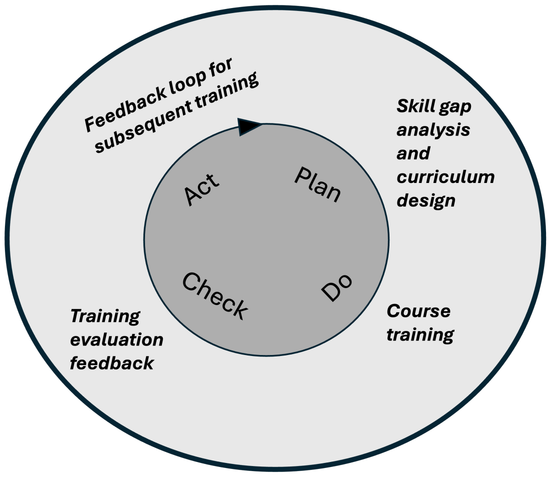
Feedback loop for subsequent training
Skill gap analysis and curriculum design
Act
Plan
Check
Do
Training evaluation feedback
Course training

Figure 1. PDCA cycle for the process activities of the present study (Adapted from Schönsleben (2023))

3.2. Data collection

To establish the knowledge and skill gaps in software safety in Tanzania, an online survey of software professionals from the public and private sectors was conducted from September to October 2022, focusing on assessing the knowledge and skills of software professionals on various aspects of software safety, and whether and how they practice software safety. Specifically, informed by software safety knowledge and skill areas covered by the work of Kang and Do (2021), a questionnaire was developed that focused on establishing the respondents' awareness of and prior training on software safety; knowledge, skills and use of best practices when developing safety-critical systems; understanding and use of software safety standards, and the domains in which safety-critical systems are developed.

The survey of software professionals was followed by in-depth semi-structured interviews, to ascertain what had been reported in the survey. The follow-up interviews were conducted with 25 software professionals from 70 software professionals who had responded to the survey: the 25 interviewees were purposely sampled from five regions of Tanzania (Dar es Salaam, Dodoma, Mbeya, Mwanza and Zanzibar). The respondents were shortlisted based on participating organisations' regional representations and the potential that the portfolio of software the respondents were engaged in development were safety critical. In this case, the respondents from the health domain were most targeted as the systems they engaged portrayed a greater potential for being safety-critical. Mirroring the survey questions, interview participants were asked to explain how they considered their level of knowledge/skills in general software safety-related competencies. The survey and interviews of practitioners were used to establish the software safety skill gaps in Tanzania, producing a skill gaps report that was the basis for the development of the curriculum for the professional course on software safety for software practitioners in Tanzania.

Using the developed curriculum, a series of four training workshops were conducted between June and December 2023, covering recent graduates, early career software engineers, established software professionals, and combined university lecturers and more established software professionals, respectively. Each training workshop was embedded with training evaluation to capture trainees' feedback, which was propelled for improvement of the training delivery in the subsequent training workshop. The training experience of participants and the teaching strategies on the different topics were analyzed to foster the training feedback loop.

3.3. Data analysis

A total of 87 software professionals in Tanzania responded to the online survey on the software safety skill gaps analysis. A preliminary review of responses to the survey revealed that a total of 70 respondents had answered the key questions of the survey as well as the respondents' demographics and other personal information. The 17 respondents who had provided demographic information without answering key questions related to software safety were excluded from further analysis. It is the responses from 70 respondents that were further analysed to establish the skill gaps on software safety in Tanzania. Moreover, the 25 follow-up interviews were transcribed, producing transcripts that were analysed thematically to complement the survey results.

The thematic analysis process involved coding and analysis based on Clarke and Braun (2017) guidelines for using thematic analysis. The authors familiarized with the data reading the transcripts several times, and in turn the transcripts were read line-by-line and initial codes written in a column alongside the transcripts. The authors independently coded a partitioned subset of the interviews for purposes of reflexivity, providing stimulating alternative standpoint. The three standpoints were cross-referenced to notice and reflect on the authors differences of viewpoint.

The independently developed codes were jointly harmonized to obtain an initial complete coding. The developed codes were categorized to list the codes and group them by similarity (Clarke and Braun, 2017; Patton, 1987; Glaser and Strauss, 2017). An inductive, data-driven approach was used in the categories and the entire data set to synthesize the emergent themes as guided by Braun and Clarke (2021). Table 1 and Table 2 represent the result of the application of thematic analysis process to a core section of data in which the need for professional training and update of the curriculum in tertiary institutions on software safety and institutional strengthening to regulate the safety aspects of technological systems is synthesized.

The analysis of data from the survey and interviews enabled us to establish the knowledge and skill gaps for software safety in Tanzania. This gave us a skill gaps analysis report. A review of the skill gaps analysis report was conducted through a stakeholders' workshop to ascertain and validate the software safety skill gaps in Tanzania. The validated skill gaps were used by subject matter experts for the design of the curriculum that aimed to address the identified software safety training needs. The draft of the curriculum was also reviewed by stakeholders, facilitating its improvement. The revised curriculum was then used to guide the development of training materials and the actual training of software practitioners. We analysed both the quantitative and qualitative feedback from participants, collected at the end of each day of the four training workshops of five days, to gauge the level of understanding for each topic and obtain any specific feedback from trainees. The analyzed feedback for each day was used to inform the training on the next day and or the next training workshop.

3.4. Ethical considerations

The study was approved by the University of Dar es Salaam in Tanzania, and participation in the study was voluntary, enabling participants to withdraw at any stage in the study. In addition, the identities of participants are anonymized.

Table 1. Transcript codes

S/N	Transcript	Codes
1	"Most organisations have been focusing on security of software rather than safety it is high time to bring on board this knowledge to organisations"	Security versus safety knowledge
2	"I believe this is a very ignored aspect in software development"	safety safety consideration
3	"High level institutions should consider incorporate into curriculum"	safety curriculum
4	"By prioritising safety training to Tanzanians, we may be able to identify, estimate and by so be able to prevent hazards"	safety training hazard
5	"Safety should be prioritised, and this is possible if the systems development practitioners are trained well to identify safety issues and develop safety procedures to handle and improve the system safety"	safety systems development
6	"There should be regular education about this issue to promote greater understanding among developers"	safety education
7	"Tanzania is moving fast in terms of technology adaptation, software safety practices become crucial now and then. I think we as tech/software engineers need to consider this practice while developing software."	safety technology adaptation

Table 2. Code categories

Category	Codes
safety practice	security versus safety safety consideration knowledge technology adaptation Systems development
Safety skills and knowledge	security versus safety safety hazard training knowledge
Safety stakeholders	curriculum education technology adaptation training

4. Results

This section presents the results, which are then discussed in Section 5.

4.1. Skills required to develop safety-critical systems in developing countries

This section provides results from the needs assessment survey and interviews conducted with various software engineering stakeholders in Tanzania. Additionally, we also describe the design of the curriculum based on the findings of the needs assessment as well as design decisions made on the curriculum design.

4.1.1. Demographics of survey respondents

To design an effective curriculum for software safety professional training, we envisioned potential trainees by analysing the needs assessment survey with 87 respondents. The age profile of the survey participants indicates a predominantly young demographic in the software engineering sector, with the majority, 52.9%, falling within the 26-35 age range. This is mainly because the ICT industry in general including software engineering is a very young field in Tanzania. Note that the first computer science department in the country was established in 1999 at the University of Dar es Salaam. Our analysis showed that the software engineering (SE) sector is also dominated by individuals with tertiary education: about 90% of respondents have at least a bachelor's degree, with only 10% at the diploma level and there were no certificate holders. This suggests a job market heavily employs university graduates. Regarding the employment industry, 40% of respondents worked in organisations where software engineering is the core business ("SE specialising"), while 36% were employed in institutions where SE is not the core business. The remaining professionals were either independent consultants or unemployed. In terms of functional areas within the SE industry, our findings show a diverse range of practices. Most professionals are involved in multiple functions. The most prevalent areas include software programming, systems analysis, software project management, and software design/architecting, indicating the development of pure software applications rather than embedded systems. On the other hand, roles like software qualification, system audit, and hardware programming are less prominent. Table 4.1.1 summarizes the demographics of survey respondents.

Table 3. Demographics of survey respondents

Profile Item	Category	Percentage
Age	18-25	9.2%
	26-35	52.9%
	36-45	26.4
	46-55	10.3%
	55+	1.1%
Sex	Male	92%
	Female	8%
Highest Education	Certificate	-
	Diploma	10.3%
	Bachelor Degree	57.3%
	Masters	27.6%
	PhD	4.6%
Employment Status	Employed by SE specialising Enterprise	34.5%
	Employed by non-SE specialising Enterprise	41.4%
	Independent/Consulting Software Engineer	13.8%
	Unemployed	10.3%
Length worked in the software engineering field	Less than 1 year	5.7%
	1-2 years	14.9%
	3-4 years	28.7%
	5-10 years	31%
	10+ years	19.5%
Functional areas of software engineering	Software programming	51.7%
	Systems analysis	49.4%
	System audit	21.8%
	Software design/architecting	39.1%
	Software project management	43.7%
	Hardware programming	14.9%
	Software qualification	16.1%

4.1.2. Awareness of software safety

The survey asked participants about their awareness of software safety concepts. As shown in Figure 2 and Figure 3, the respondents showed moderate awareness with most safety concepts scoring an average of at least 3 on a scale from 1- 5 where 1 indicates No knowledge/skill and 5 indicates Fully knowledgeable/skilled – no/very little development required. On the contrary, follow up interviews indicated that most practitioners interviewed (21 out of 24) had no prior knowledge on software safety and most of them confused it with software security, that is why they rated themselves very highly in the survey. These insights were crucial for tailoring our training program to the current landscape of software engineering professionals in Tanzania.

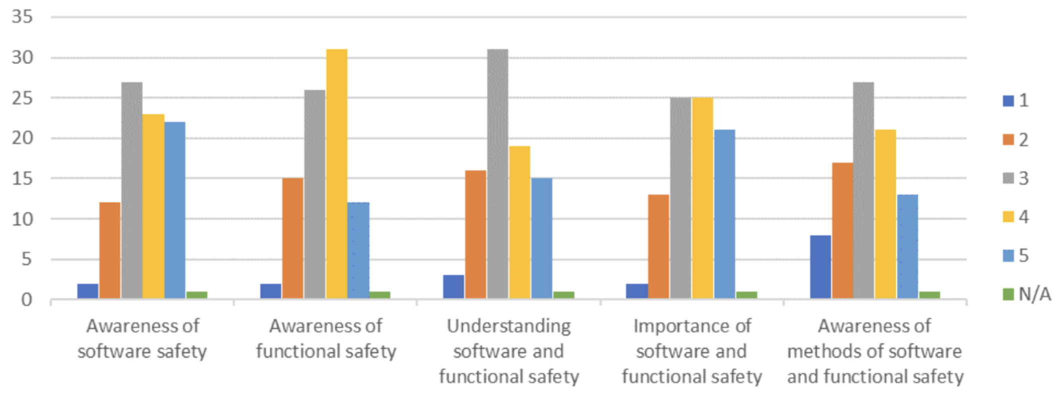

Figure 2. Awareness of functional safety

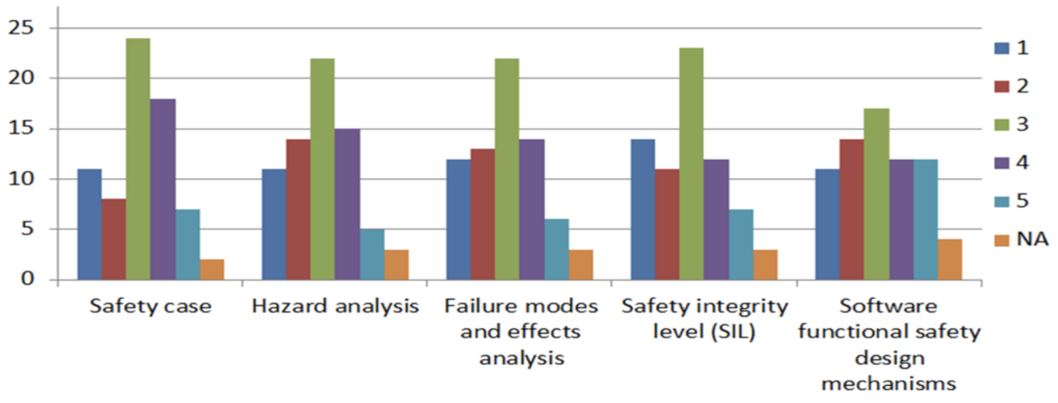

Figure 3. Awareness of software safety concepts

4.1.3. Decisions for the curriculum

Based on the findings from the survey, interviews and stakeholders workshop, the curriculum development team made the following decisions precisely focusing on the context of Tanzania.

- Since the majority of the stakeholders confused software safety with software security, our curriculum covers a wide range of introductory topics on safety and functional safety. It clearly explains the similarities and differences between software safety and security, with the aim of correcting common misunderstandings.
- This study found that in Tanzania, software-only systems are much more common than embedded systems. For instance, in the health sector hospitals use electronic medical record (EMR) systems and laboratory information systems to manage patient care, which, if not well-maintained, could pose safety risks. Similarly, in the water supply sector, software is used to monitor and/or control the process of treating water before it is distributed to homes and offices, ensuring it is safe to drink. In the energy sector, software is used in managing electricity generation, distribution, and transmission. And in air transportation, software systems help monitor airport activities like safe take-offs and landings. Given the prevalence of software-only systems in the country over embedded systems, the curriculum was tailored towards the development of software with little to no hardware. Therefore, topics related to ensuring hardware safety were not the focus of the curriculum.
- Although internally developed embedded systems are relatively scarce, imported systems such as automated security systems (e.g., those used in airports) and automated drug dispensing systems (in hospitals) are more common. This prevalence has necessitated the inclusion of content focused on the inspection and safe deployment of such systems in the curriculum.
- The survey also revealed that most software engineering professionals in the country are at a junior level and mainly involved in basic tasks. This suggests that there is a lack of advanced software processes, similar to the lower levels of the Capability Maturity Model Integration (CMMI) (Constantinescu and Iacob, 2007). So, our curriculum also includes essential software engineering practices, such as code reviews and integration testing to help improve the quality of software development processes in general.

The developed curriculum is presented in Table 4.

4.1.4. Content Development

The content to teach the included topics and subtopics in the curriculum was developed by first leveraging available open educational resources such as the system safety graduate course offered at Massachusetts Institute of Technology (MIT) Open Courseware (2016) and then contextualising this content given the expected experiences of the potential trainees. The following are two notable cases of software safety content contextualization. First, the examples used in the training were mainly related to health since our survey of software practitioners in Tanzania shows that most of the prevailing safety critical systems are from the health domain. Second, given the dominance of digital banking and financial services in Tanzania, and mobile money services in particular, more and more financial technology (fintech) services are on the rise and so more financial mobile applications are being developed by developers within the country. Failure or vulnerability of these systems can lead to the loss of a significant amount of money, given the popularity of mobile money trading in Tanzania. To cater for developers from the financial domain, we expanded the definition of software safety to also cover substantial financial losses, in line with the mishap severity categories in the MIL-STD 882D system safety standard (Department of Defense, 1993). We, therefore added examples of failures related to fintech and mobile money systems. This made it easier to relate with the target audience. Importantly, while our initial aim was to develop content completely tailored to Tanzania, we realised that this was not possible for all cases. For instance, to get someone to relate to significant repercussions of failures of safety-critical systems, examples of where such failures have occurred, are needed. Given that such failures and their associated reasons are not announced or documented in the country yet, it was difficult to find them. For such scenarios we used popular accidents that are well known such as the Boeing 737 MAX accidents and the Toyota unintended acceleration case study (Koopman, 2014).

Table 4.: The software safety curriculum

Module	Topic	Sub-topics
Module 1: Overview of software safety	Software safety and functional safety	The concept of software safety Importance of software safety Functional safety Meaning of functional safety Functional safety Vs. Software safety
	Safety-critical software systems	Attributes (key features) of safety-critical software systems Safety-critical functions Examples of safety-critical software systems and failures of such systems Causes of failure of safety-critical software systems
	Safety Vs. Security	Security concept Relationship between safety and security
	Safety Vs. Reliability	Reliability concept Relationship between safety and reliability
	Software dependability	Concept of software dependability Relationship between software dependability and software safety
Module 2: Process for software safety	Overview	Safety as an overarching consideration in the development of safety-critical software systems Safety and software development methodologies Model-driven development
	Safety considerations in requirements engineering	Safety analysis Hazards Analysis, Failure Mode and Effects Analysis (FMEA), Fault Tree Analysis (FTA), Risks and Severity analysis Translation of hazards to safety requirements Safety requirements traceability Software safety requirements tools

	Safety considerations in the design phase	Safety design approaches and techniques Safety kernels, barriers, checks, and safety in failure (fail safe and fail soft) Functional safety design mechanisms Software safety design tools
	Safety considerations in the implementation phase	Pipeline setup (Code reviews, coding standards) Code quality metrics Model driven development Libraries and third-party plugins Software safety implementation tools Fault Tolerance Techniques N Version Programming, Recovery Blocks Other Techniques Data Redundancy
	Safety considerations in the testing and quality assurance	The V-model Equivalence partitioning, boundary value analysis, and error guessing in software safety testing Back to back software safety testing Fault injection in safety testing Code coverage in safety testing Software inspections Development of safety cases
	Safety considerations in the deployment and maintenance phase	Installation and commissioning of safety-critical software systems Safety validation in real/operational environment Management of change in safety-critical systems Retirement of safety-critical systems
Module 3: Software safety standards	Overview of software safety standards	Introduction to software safety standards Importance of software safety standards General software safety standards (IEC 61508
	Domain-specific software safety standards	Common things in the standards (Safety Integrity Level (SIL), hazard analysis, etc.) Software/System safety lifecycle in domain-specific standards Personal care robots (ISO 13482) Automotive (ISO 26262) Aviation (DO-178C) Industrial automation () Marine (ISO 17894)

	Certification and auditing	The software safety auditing process Auditing of safety requirements Auditing of safety design Product and program auditing How to get your software certified
	Liability and contractual issues	Liability of safety-critical systems Domestic and international legislations/regulations Developing safety guidelines

4.2. Experience of training professionals to develop safety-critical systems

We wanted to understand the extent to which different topics were understood by trainees, and the reasons for adequate and inadequate understanding of different topics. Table 5 shows the degree of understanding for the different topics.

Table 5.: Degree of understanding for different topics. Note that 5=Fully understand, 4=Good level of understanding, 3=Somehow understand, 2=Little understanding, 1=Did not understand, Resp=Total Respondents, T= Total for 5 to 1, OU=Optimal understanding (sum of 5 and 4), UNU=Unoptimal or no Understanding (sum of 3, 2 and 1).

Sn	**Topic**	**Resp**	**5**	**4**	**3**	**2**	**1**	**T(%)**	**OU (%)**	**UNU (%)**
1	Model Driven Development	67	43.3	46.3	9.0	1.5	0.0	100.0	89.6	10.4
2	Hazard Analysis	69	58.0	31.9	7.2	1.4	1.4	100.0	89.9	10.1
3	Failure Mode Effect Analysis	69	34.8	39.1	23.2	1.4	1.4	100.0	73.9	**26.1**
4	Failure Mode Effect Criticality Analysis	69	42.0	39.1	15.9	1.4	1.4	100.0	81.2	18.8
5	Fault Tree Analysis	67	32.8	40.3	22.4	4.5	0.0	100.0	73.1	**26.9**
6	Safety Integrity Levels	69	17.4	50.7	27.5	4.3	0.0	100.0	68.1	**31.9**
7	Software Traceability	69	44.9	44.9	8.7	1.4	0.0	100.0	89.9	10.1
8	Simplicity	51	54.9	33.3	7.8	3.9	0.0	100.0	88.2	11.8
9	Decoupling	52	46.2	40.4	11.5	1.9	0.0	100.0	86.5	13.5
10	Reducing opportunity for human errors	52	63.5	30.8	5.8	0.0	0.0	100.0	94.2	5.8
11	Design for controllability	52	42.3	50.0	7.7	0.0	0.0	100.0	92.3	7.7
12	Safety kernels	53	35.8	39.6	22.6	1.9	0.0	100.0	75.5	**24.5**
13	Checks/monitors	53	50.9	34.0	13.2	1.9	0.0	100.0	84.9	15.1
14	Fail-safe design (passive)	53	52.8	32.1	15.1	0.0	0.0	100.0	84.9	15.1

15	Fail-safe design (active)	52	44.2	40.4	15.4	0.0	0.0	100.0	84.6	15.4
16	Damage reduction	53	50.9	37.7	11.3	0.0	0.0	100.0	88.7	11.3
17	Design barriers (lock-in, lock-out, and interlock)	53	47.2	30.2	18.9	3.8	0.0	100.0	77.4	**22.6**
18	Failure minimization (safety factor/margin and redundancy))	53	54.7	32.1	11.3	1.9	0.0	100.0	86.8	13.2
19	Coding standards	33	75.8	15.2	9.1	0.0	0.0	100.0	90.9	9.1
20	MISRA C++	33	63.6	24.2	9.1	3.0	0.0	100.0	87.9	12.1
21	Code review	53	58.5	32.1	7.5	1.9	0.0	100.0	90.6	9.4
22	Formal static analysis	53	49.1	37.7	11.3	1.9	0.0	100.0	86.8	13.2
23	Static analysis	54	46.3	40.7	13.0	0.0	0.0	100.0	87.0	13.0
24	Fault-tolerant programming techniques	54	51.9	35.2	11.1	1.9	0.0	100.0	87.0	13.0
25	Testing types	54	68.5	25.9	5.6	0.0	0.0	100.0	94.4	5.6
26	Equivalence partitioning	55	45.5	38.2	16.4	0.0	0.0	100.0	83.6	16.4
27	Boundary Value Analysis	54	57.4	33.3	7.4	1.9	0.0	100.0	90.7	9.3
28	Error guessing	55	61.8	32.7	5.5	0.0	0.0	100.0	94.5	5.5
29	Back to back testing	55	50.9	36.4	10.9	1.8	0.0	100.0	87.3	12.7
30	Fault injection	54	66.7	27.8	3.7	1.9	0.0	100.0	94.4	5.6
31	Coverage-based testing	55	56.4	34.5	7.3	1.8	0.0	100.0	90.9	9.1
32	Fault-based testing	55	67.3	25.5	5.5	1.8	0.0	100.0	92.7	7.3
33	Statement coverage	54	48.1	40.7	11.1	0.0	0.0	100.0	88.9	11.1
34	Branch coverage	55	50.9	38.2	7.3	3.6	0.0	100.0	89.1	10.9
35	Modified Condition/Decision Coverage (MC/DC)	55	36.4	43.6	14.5	5.5	0.0	100.0	80.0	**20.0**
36	Safety cases	34	52.9	38.2	8.8	0.0	0.0	100.0	91.2	8.8
37	Software safety standards	29	51.7	41.4	6.9	0.0	0.0	100.0	93.1	6.9
38	Generic safety standards	28	39.3	53.6	7.1	0.0	0.0	100.0	92.9	7.1
39	Commonalities across standards	29	37.9	48.3	10.3	3.4	0.0	100.0	86.2	13.8
40	Liability and legal issues	29	51.7	37.9	10.3	0.0	0.0	100.0	89.7	10.3
41	Installation and commissioning of safety-critical systems	30	60.0	33.3	3.3	3.3	0.0	100.0	93.3	6.7
42	Safety validation	30	63.3	30.0	3.3	3.3	0.0	100.0	93.3	6.7
43	Operation and maintenance	30	66.7	23.3	6.7	3.3	0.0	100.0	90.0	10.0
44	Management of change	30	60.0	26.7	13.3	0.0	0.0	100.0	86.7	13.3
45	Decommissioning and disposal	30	56.7	30.0	13.3	0.0	0.0	100.0	86.7	13.3
46	Human factors	29	58.6	34.5	3.4	3.4	0.0	100.0	93.1	6.9

The quantitative analysis of the training feedback shows that most of the topics were well understood by training participants. To mitigate the possible validity threats due to self-reporting data, the understanding of various topics had to be confirmed through group and individual activities, as well as the comprehensive quiz given at the end of each face-to-face training workshops. The analysis of the qualitative feedback from the training participants suggests that most of the topics were well understood for the following reasons: use of group activities, interactive classrooms, use of examples that participants could relate to, sequencing the training topics in such a way that each topic built on top of what participants already knew, and the use of OER (Open Education Resources) such as YouTube videos in teaching and learning of some topics. The following are some of the quotes from the written feedback from participants:

- *"The session were all interesting but the group discussions make the session more live"*
- *"The group discussion was helpful and the teachers"*
- *"Real life examples"*
- *"The flow of teaching went very well, concepts built on top of each other well leading to a very good understanding"*
- *"Youtube video, simple explanation came from moderator"*

However, it can be seen from Table 5 that some topics were poorly understood by 20% (or more) of the responding trainees. Specifically, the few not-so-well-understood topics were mostly related to the analysis of safety requirements and design. For safety requirements, SIL, FTA, and FMEA recorded higher levels of less understanding compared to others, and, for safety design, safety kernels and design barriers (lock-in, lock-out, and interlock) were reported to have been less understood. MC/DC was leading among the less understood topics on safety testing. Further analysis of this situation suggested that poor understanding of some concepts could be related to the lack of relevant local examples for use when teaching these concepts and/or inadequate background in the mathematics required to properly understand some of these concepts (e.g., SIL).

It was also learned through the analysis of the qualitative feedback from trainees that future training could consider the development of an example safety-critical system during the training, to put what is taught into practice. So, finding one such software example from the context of a developing country and working towards building it during the training period could reinforce teaching and learning of the development of safety-critical systems in developing countries. Relatedly, the feedback from trainees also showed that some topics were taught using imaginary examples that were not drawn from the context of developing countries. Other topics were taught using hardware (instead of software) examples that were drawn from the context of developed countries. For example, while examples of software that run in vehicles or aircraft might be easy to use in teaching and learning the skills to develop safety-critical systems, such examples can be hard to follow by software developers from developing countries, which hardly manufacture vehicles or aircraft. This in turn complicates the process of teaching and learning these concepts. The trained software practitioners also emphasised that software (instead of hardware) examples from the context of developing countries would be most appropriate for teaching and learning.

Besides, the qualitative feedback from trainees shows that there were more training materials covered within five days of face-to-face training, which might have hindered the ability of trainees to follow and retain all the materials covered. Coverage of more materials in limited time was especially because of limited project funding, which could only cover four face-to-face training workshops of five days each.

5. Discussion

5.1. Curriculum for software safety in developing countries

From the needs analysis phase, we found that results from the survey and the follow up interviews were contradicting due to the confusion between safety and security. We envision that this is a common practice where practitioners feel the need to rate themselves high when it comes to topics in their own domain, even though they are not well familiar with them (Wetzel et al., 2016). We therefore recommend that, to clearly identify a knowledge/skills gap, especially for curriculum development, data triangulation is important. One should not only rely on surveys, but interviews and observation should also be conducted whenever possible. The lack of awareness on software safety concepts and tendency of confusing safety and security is attributed to the fact that both undergraduate and post graduate curricula of the different universities in Tanzania do not include software safety topics. On the other hand, security has been a buzz word and therefore security topics are widely taught within the country. This specific context therefore called for a curriculum that not only defines software safety but also distinguishes it clearly from software security.

The nature of the software engineering domain in Tanzania is characterised by a youthful demographic and professionals working in small software development firms or large firms whose main businesses are not software development. By adding key software engineering methods like code reviews and integration testing to the curriculum, we are not just aiming to promote awareness of software safety but also improve the overall software development quality in Tanzania. Case studies such as that of the Toyota unintended acceleration (Koopman, 2014) reveal that even failing to follow basic coding practices such as code review can lead to catastrophic events. However, this required a balance between making sure that trainees understand what is being taught from software engineering concepts and ensuring that enough safety topics have been taught.

Given that the nature of developing countries is similar in terms of social and economic factors, we envision that the software industry in other developing countries may also be similar characterised by safety critical software-only systems and imported embedded safety critical systems or may require other contextualisation, as suggested by Osman (2012). The curriculum we propose should therefore be taken as a starting point and tailored given the needs of the target trainees.

5.2. Teaching techniques for better understanding of software safety in developing countries

It can be seen from the training feedback (Section 4.2) that context-sensitivity and adult learning techniques are the most suited techniques for teaching software safety to professional software engineers in Tanzania and, by extension, developing countries. Context sensitivity is particularly emphasized by the need to use examples that trainees can relate to, as argued by Osman and Dias-Neto (2014); Osman (2012) and Fendler and Winschiers-Theophilus (2010). However, we learned through the present study that it may not always be possible to get context-sensitive examples of safety-critical systems for use in teaching each software safety concept or technique in developing countries due to lack of documentation of such systems in developing countries. Such situations might necessitate careful use of examples of software for safety-critical systems like vehicles and aircraft that are commonly created in developed countries. To address this challenge, future researchers or trainers working in the context of developing countries should try as much as possible to draw more concrete examples from the context of developing countries. A systematic mapping of safety-critical systems available in a developing country of interest might offer an appropriate pool of examples for use in training developers of safety-critical systems in the respective developing countries.

Furthermore, capitalizing on learners' prior experiences and the use of group discussions with associated feedback from facilitators are some of the ways in which adult learning techniques when training professional software engineers (Ellis and Hislop, 2004) were put into practice, increasing the understanding of the subject matter. In line with what was recommended by Galster et al. (2018), the use of YouTube videos facilitated a better understanding of some software safety concepts, e.g., Safety Integrity Levels. Thus, the importance of context-sensitivity and adult learning techniques when teaching software safety to professional software engineers cannot be overemphasized. In short, the findings of the present study agree with the existing literature on how to teach various topics to professional software engineers.

However, different from previous studies on software safety training (see, for example, Kang and Do (2021)) which only went as far as offering recommendations on what to include in a general software safety curriculum, the present study not only offers training topics recommendations that are sensitive to the context of developing countries, but it also offers the empirical lessons on the actualization of the recommended topics on software safety, enabling, among other things, a better understanding of what works and what does not work when conducting software safety training for software professionals in the developing world.

Moreover, our analysis shows that the less understood topics were truly new to trainees and represent some of the core skills required to develop safety-critical systems. We learned that the techniques covered in the least understood topics such as SIL, FTA, FMEA, design barriers, safety kernels, and MC/DC are rare in the ICT curricula in higher learning institutions in Tanzania, and so most of the trainees were new to the development of embedded systems. Thus, to equip safety-critical systems development skills to university graduates in Tanzania and other countries similar to it, each undergraduate ICT curriculum in higher learning institutions should consider coverage of introductory topics in embedded systems. This is especially important because professional software developers who are involved (or could potentially be involved) in the development of safety-critical systems in these countries come from all ICT graduates, irrespective of their degree name or specialization. Additionally, to the best of our knowledge, there are no specialised programs in the master level focusing on development of safety-critical systems.

In addition, the math thinking in some of the least understood topics might also have influenced the degree of understanding of such topics by trainees. The FTA technique, for example, requires a proper understanding of logic gates, which might have been a hindrance for some trainees. Similarly, SIL involves the computation of probabilities, which might be tricky for some trainees. Therefore, to produce more graduates with the ability to properly learn, master and get involved in the development of safety-critical systems, the university curricula for different ICT degrees in developing countries need to put a strong emphasis on discrete mathematics and statistics.

Finally, because software safety concepts are new for most software professionals from developing countries, care must be taken regarding the amount of time to cover the training materials. If there is limited time, good-quality coverage of a few concepts–especially the ones at the core of software safety–is preferable. More concepts can be covered if time allows. However, covering more materials within a limited time might lead to low retention, defeating the whole purpose of a practical-oriented professional training course on software safety. A self-paced course on the development of safety-critical systems could be a way to balance between the limited resources of time and money and the amount of training materials covered. This is why we also developed a self-paced course for that purpose. Importantly, to get more time for coverage of more training materials through face-to-face training, it might be important to increase the amount of funds available for the training. This could involve making trainees pay more money for the training, different from the arrangements reported in the present paper, in which trainees were not charged to attend the training.

5.3. Contribution and Implications

The present paper makes an empirical contribution by providing a detailed account of designing and implementing a software safety curriculum for professional software engineers in Tanzania and, by extension, other similar developing countries. This contribution has implications for both research and domain practices. For researchers, this empirical contribution implies opportunities to theorize about the development of safety-critical systems in developing countries, producing theories for better analysis, explanation, description, and prediction of the development of safety-critical systems in developing countries. For instance, software engineering researchers could develop frameworks, taxonomies, and models about the nature of safety-critical systems in developing countries, how such systems should be developed or procured in developing countries, and training and regulatory requirements for safety-critical systems in developing countries. Among other things, researchers could develop models to help software practitioners in developing countries to automatically identify safety-critical aspects of the systems they develop, so that appropriate safety considerations can be made during development. For practitioners, the empirical contribution of the present paper implies opportunities for benchmarking key considerations for improving how safety-critical systems are developed, procured, and regulated in developing countries. For instance, education and training institutions can use the empirical account in the present paper to improve aspects of education and training curricula related to the engineering of safety-critical systems. Software development teams could consider using the V-model whenever they are developing a system that is regarded to be safety-critical. This could require review and strengthening of regulatory practices for safety-critical systems in developing countries.

6. Conclusion and recommendations

Like the rest of the world, Tanzania as a developing country has been experiencing an increase in the use of cyber-physical systems, i.e., systems with software-controlled hardware, some of which are developed by software engineers in Tanzania. Engineering of cyber-physical systems calls for special consideration of safety because any safety violation could cause injuries, deaths, and or damage to property and the environment. However, the knowledge and skills to develop such safety-critical systems are almost non-existent in developing countries like Tanzania, due to little or no emphasis of such competencies in the university ICT curricula in most developing countries. The existing efforts to train software practitioners on software safety are not sensitive to the context of developing countries and have only proposed the topics to be included in the software safety curriculum, without actualizing such a curriculum, lacking empirical lessons on the design and actualization of software safety curriculum in developing countries.

The present paper provides a detailed empirical account of designing and implementing a software safety curriculum for professional software engineers in Tanzania, drawing lessons that could also be applied to other developing countries. We particularly underscore the importance of context-sensitive content and pedagogy when teaching software safety to practitioners in developing countries. We also recommend a systematic mapping of examples of safety-critical systems from a developing country of interest to design and deliver a context-sensitive software safety curriculum. The implications of our contribution for research and practice are also provided. As regards the skills required to develop safety-critical systems in the context of developing countries, we found that most of what is required are what can be regarded as basic software safety skills, because, although they are basic, these skills can still be in limited supply in developing countries, due to limited or no coverage in university curricula, apparently lenient regulatory environment for safety-critical systems in developing countries, and more dependence of safety-critical systems developed in developed countries. The ability of software development teams in developing countries to identify the safety-criticality of systems and deploy appropriate development practices (e.g., the application of the V-model) could improve how safety-critical systems are developed in developing countries. We also found that the teaching strategies that are most suited for teaching different topics on software safety in the context of developing countries are the ones that are sensitive to the local context. Specifically, the use of local examples of safety-critical systems could improve understanding of safety considerations when developing safety-critical systems in developing countries.

Although the findings of this study could be applicable to other developing countries, given the social and economic similarities among developing countries, future studies should consider replicating our curriculum in other developing countries to generate even more context-sensitive software safety-related lessons for the developing world. In particular, it might be important to compare and contrast the experiences of designing and delivering a software safety program in different developing countries. Also, relying on self-reporting data from trainees could threaten the validity of the findings of the present study. To mitigate the effects of this threat, we corroborated the evidence from self-reporting data with follow-up questions and activities. However, in the future, it might be important to observe how safety-critical systems are actually developed or procured in practice, to glean more lessons on how safety-critical systems are developed in developing countries, the skills required, and the best ways to impart such skills among software practitioners.

Acknowledgement(s)

We particularly acknowledge the Royal Academy of Engineering, UK, for funding this work through grant number ESMN 2123\2\132. Anonymous participants in this study are also immensely acknowledged.

Disclosure statement

The authors declare that there is no known competing interest.

Funding

This work was funded by the Royal Academy of Engineering, UK, through grant number ESMN 2123\2\132.

7. References

References

M. Ardis and N. R. Mead. The development of a graduate curriculum for software assurance. 2011.

L. P. Binamungu, S. Maro, G. Justo, and J. Ndanguzi. Assessing software safety knowledge and skill gaps in Tanzania. In *2024 IST-Africa Conference (IST-Africa)*, pages 1–8. IEEE, 2024.

E. Blake. Software engineering in developing communities. In *Proceedings of the 2010 ICSE workshop on cooperative and human aspects of software engineering*, pages 1–4, 2010.

E. H. Blake and W. D. Tucker. Socially aware software engineering for the developing world. 2006.

V. Braun and V. Clarke. One size fits all? what counts as quality practice in (reflexive) thematic analysis? *Qualitative research in psychology*, 18(3):328–352, 2021.

J. M. Case, D. M. Fraser, A. Kumar, and A. Itika. The significance of context for curriculum development in engineering education: a case study across three african countries. *European journal of Engineering education*, 41(3):279–292, 2016.

V. Clarke and V. Braun. Thematic analysis. *The journal of positive psychology*, 12(3):297–298, 2017.

R. Constantinescu and I. M. Iacob. Capability maturity model integration. *Journal of Applied Quantitative Methods*, 2(1):31–37, 2007.

Department of Defense. Standard practice for system safety, 1993. URL `https://mail.system-safety.org/Documents/MIL-STD-882D.pdf`.

H. J. Ellis and G. W. Hislop. Techniques for providing software engineering education to working professionals. In *34th Annual Frontiers in Education, 2004. FIE 2004.*, pages F1C–19. IEEE, 2004.

J. Fendler and H. Winschiers-Theophilus. Towards contextualised software engineering education: an african perspective. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 599–607, 2010.

M. Galster, A. Mitrovic, and M. Gordon. Toward enhancing the training of software engineering students and professionals using active video watching. In *Proceedings of the 40th international conference on software engineering: Software engineering education and training*, pages 5–8, 2018.

B. Glaser and A. Strauss. *Discovery of grounded theory: Strategies for qualitative research.* Routledge, 2017.

J.-W. Kang and S.-R. Do. The needs analysis of software safety education program for common competency area. *Journal of Information Processing Systems*, 17(5), 2021.

A. Kawano, Y. Motoyama, and M. Aoyama. A lx (learner experience)-based evaluation method of the education and training programs for professional software engineers. In *Proceedings of the 2019 7th International Conference on Information and Education Technology*, pages 151–159, 2019.

K. Kobata, T. Uesugi, H. Adachi, and M. Aoyama. A curriculum development methodology for professional software engineers and its evaluation. In *2014 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*, pages 480–487. IEEE, 2014.

P. Koopman. A case study of toyota unintended acceleration and software safety. *Presentation. Sept*, 2014.

Massachusetts Institute of Technology (MIT) Open Courseware. 16.863j system safety, spring 2016, 2016. URL `https://ocw.mit.edu/courses/16-863j-system-safety-spring-2016/`.

M. Moore. What is industry 4.0? everything you need to know. *Ang. A: TechRadar*, 2019.

R. Osman. Teaching software engineering in developing countries: A position paper. In *2012 IEEE 36th Annual Computer Software and Applications Conference*, pages 648–653. IEEE, 2012.

R. Osman and A. C. Dias-Neto. Motivating by examples: An empirical study of teaching an introductory software engineering course in brazil. In *2014 IEEE 38th Annual Computer Software and Applications Conference*, pages 245–250. IEEE, 2014.

M. Q. Patton. How to use qualitative methods in evaluation. *Saint Paul*, 1987.

R. Rafeh and A. Rabiee. Towards the design of safety-critical software. *Journal of applied research and technology*, 11(5):683–694, 2013.

V. Ruwodo, A. Pinomaa, M. Vesisenaho, M. Ntinda, and E. Sutinen. Enhancing software engineering education in africa through a metaversity. In *2022 IEEE Frontiers in Education Conference (FIE)*, pages 1–8. IEEE, 2022.

P. Schönsleben. 18.2.1 the deming cycle (pdca cycle) and the shewhart cycle, 2023. https://opess.ethz.ch/course/section-18-2/18-2-1-the-deming-cycle-pdca-cycle-and-the-shewhart-cycle/. Last accessed: 1st February 2024.

A. Striuk and S. O. Semerikov. Professional competencies of future software engineers in the software design: teaching techniques. In *Journal of Physics: Conference Series*, volume 2288, page 012012. IOP Publishing, 2022.

M. K. Stroud and T. Pfitzer. The evolution of continuing education & training for safety & mission assurance professionals. *Journal of Space Safety Engineering*, 3(2):64–66, 2016.

L. Thomas, P. Waterson, and S. Trapp. Eight years of delivering professional education and training for software engineering at fraunhofer iese: An experience report. In *19th Conference on Software Engineering Education & Training (CSEET'06)*, pages 131–140. IEEE, 2006.

I. Warren. Teaching patterns and software design. In *Proceedings of the 7th Australasian conference on Computing education-Volume 42*, pages 39–49, 2005.

E. Wetzel, J. R. Böhnke, and A. Brown. Response biases. 2016.

H. Winschiers-Theophilus. Cultural appropriation of software design and evaluation. In *Handbook of research on socio-technical design and social networking systems*, pages 699–710. IGI Global, 2009.

A. Zeid. Lessons learned from establishing a software engineering academic programme in developing countries. In *20th Conference on Software Engineering Education Training (CSEET'07)*, pages 11–18, 2007. .