Empirical Software Engineering (2024) 29:90
https://doi.org/10.1007/s10664-024-10472-6

Adoption of automated software engineering tools and techniques in Thailand

Chaiyong Ragkhitwetsagul[1] · Jens Krinke[2] · Morakot Choetkiertikul[1] · Thanwadee Sunetnanta[1] · Federica Sarro[2]

Accepted: 6 March 2024 / Published online: 10 June 2024

Abstract
Readiness for the adoption of Automated Software Engineering (ASE) tools and techniques can vary according to the size and maturity of software companies. ASE tools and techniques have been adopted by large or ultra-large software companies. However, little is known about the adoption of ASE tools and techniques in small and medium-sized software enterprises (SSMEs) in emerging countries, and the challenges faced by such companies. We study the adoption of ASE tools and techniques for software measurement, static code analysis, continuous integration, and software testing, and the respective challenges faced by software developers in Thailand, a developing country with a growing software economy which mainly consists of SSMEs (similar to other developing countries). Based on the answers from 103 Thai participants in an online survey, we found that Thai software developers are somewhat familiar with ASE tools and agree that adopting such tools would be beneficial. Most of the developers do not use software measurement or static code analysis tools due to a lack of knowledge or experience but agree that their use would be useful. Continuous integration tools have been used with some difficulties. Lastly, although automated testing tools are adopted despite several serious challenges, many developers are still testing the software manually. We call for improvements in ASE tools to be easier to use in order to lower the barrier to adoption in small and medium-sized software enterprises (SSMEs) in developing countries.

Keywords Automated software engineering · Empirical study · Online survey · Small and medium-sized software enterprises

1 Introduction

Automated Software Engineering (ASE) is the use of software tools and techniques to automate the tasks of analysis, design, implementation, testing, and maintenance of software

Communicated by: Emerson Murphy-Hill

✉ Morakot Choetkiertikul
morakot.cho@mahidol.ac.th

Extended author information available on the last page of the article systems to achieve significant improvements in quality and productivity. ASE tools and techniques have been adopted by large or ultra-large software companies (Ayewah et al. 2007, 2008; Morgenthaler et al. 2012; Johnson et al. 2013; Ståhl and Bosch 2013; Kochhar et al. 2015; Memon et al. 2017; Petrović and Ivanković 2018; Petrovic et al. 2018; Winters et al. 2020; Petrovic et al. 2022) and addressed their problems. However, they might not be effective for small and medium-sized enterprises (SMEs) due to different contexts and constraints (Yang 2022). Large or ultra-large software companies usually have more resources, which can allow them to invest in training, research and development, etc. SMEs often have fewer resources (Van de Vrande et al. 2009; Peças and Henriques 2006), which can make it more challenging for them to adopt advanced or even contemporary ASE tools and techniques. They may still use practices from their legacy software or they may have different needs from their target users and development teams, compared to large or ultra-large companies (Yang 2022). Furthermore, the research community still needs to better support software developers (e.g., reducing rework, improving timely delivery, and managing unintentional complexity) in any sizes of software companies and automated tool support is critical for this (Ozkaya 2022).

Software companies in developing countries are often almost exclusively small and medium-sized enterprises (SMEs). Since there is no widely accepted criterion, we decided to follow the definition of SMEs by the European Commission (Commission European 2020). A company will be considered as small or medium-sized when it has fewer than 250 employees and has an annual turnover of less than €50m or an annual balance sheet total of less than €43m, and vice versa for large and ultra-large companies. Thailand has a growing software industry with a $4 billion software market value in 2020 and a steady growth year by year (Digital Economy Promotion Agency 2021). A survey and assessment of the Thai digital industry's software segment in 2022 from the Digital Economy Promotion Agency (DEPA), Thailand, reports that software sector has over 13,000 companies and more than 140,000 software developers and shows that more than 99% of companies in Thailand are SMEs (12,925 out of 13,013 surveyed companies) (Digital Economy Promotion Agency 2022).

Thailand's software companies are facing many challenges including configuration management, quality assurance, and project assessment and control (Sunetnanta et al. 2016). Moreover, software companies in Thailand still employ traditional software development approaches and rely heavily on manual tasks performed by developers, which is costly and tedious. Moreover, based on our experience working with Thai SSMEs, we observed that some of the companies are mainly working on small and short-lived projects, which likely to prevent them from adopting ASE tools because the benefits gained from adopting the tools is low. Our previous case study (Ragkhitwetsagul et al. 2022) with four Thai Software SMEs (SSMEs) showed that the studied companies are facing challenges in adopting contemporary software best practices, which are instead often well-adopted in large or ultra-large software companies. The faced challenges include, among others, lack of testing (including automated testing) and code-related issues (including lack of code analysis and measurements). Such challenges can be mitigated or partially addressed by adopting contemporary ASE tools and techniques. However, the lack of testing not only affects the quality of the software, but it also prevents the adoption of advanced practices and Automated Software Engineering tools and techniques.

There are studies on the adoption of a specific tool category. For example, static code analysis in large software companies (Johnson et al. 2013) or open-source projects (Vassallo et al. 2018; Tomasdottir et al. 2020); testing of Android apps from open-source projects and Windows apps from Microsoft (Kochhar et al. 2015); test automation in 25 countries, mainly in Europe (Wang et al. 2020); and continuous integration in open-source projects (Hilton et al. 2016) or industry (Ståhl and Bosch 2013). However, there has been no study in the context of developing countries and Asian cultures, such as Thailand. Thus, we want to fill the gap by studying to what extent contemporary ASE tools and techniques are adopted in practice by software developers in Thailand and what are the challenges in ASE tools and techniques adoption.

For that reason, we have conducted a survey focusing on software engineering challenges faced by SSMEs in Thailand when adopting and using ASE tools and techniques, including software measurement, static code analysis, continuous integration and software testing.

The contribution of this paper is *a large survey on the adoption and usage of ASE tools and techniques by Thai software developers working in Thailand's growing software industry, mainly consisting of SSMEs.* Our survey allows the observation of the challenges that Thai software developers face when automating software engineering tasks like software measurement, static code analysis, continuous integration and automated testing. Moreover, this contribution includes an observation of how different groups of participants have different levels of experience and different levels of adoption and usage of ASE tools and techniques. This analysis also allowed us to observe whether different levels of experience lead to more or less significant problems when automating software engineering tasks.

The next two sections present the methodology and the results of our study, followed by a section on lessons learned. Related work is discussed in Section 5, followed by a discussion of threats to validity and conclusions.

2 Methodology

Our previous work (Ragkhitwetsagul et al. 2022) on identifying software engineering challenges in small and medium-sized software enterprises (SSMEs) used semi-structured interviews with four Thai SSMEs. We interviewed 20 software developers and managers regarding their day-to-day software development routines and identified the challenges they were facing. After performing coding of the interview transcripts and applied thematic analysis, we found the following two main common challenges.

- **Lack of testing** including no automated or unit testing, low test quality, and lack of testing knowledge and time.
- **Code-related issues** such as lack of code analysis and measurements, low code quality, and coding convention issues.

This previous work provides the motivation for the current study. Although it gives insights that Thai SSMEs are still facing contemporary best practices in software engineering like software measurement, software analysis, and automated testing and continuous integration, the finding from the study is limited to only 4 Thai SSMEs that we performed the interviews. Thus, the results may not be generalized to all Thai SSMEs.

Based on the results from the previous study, we created an online survey focusing on the two main identified software engineering challenges faced by SSMEs in Thailand including software measurement, static code analysis, and software testing, plus ASE tools and techniques that support such challenges. We also included continuous integration in the survey since the practice is closely related to software testing. The survey results are used to observe the extent of the adoption and to understand the challenges of the adoption of ASE tools by practitioners in Thailand.

2.1 Research Questions

The study aims to answer the following research questions.

1. *RQ1: To what extent are developers in Thailand familiar with the concept of Automated Software Engineering?* We want to understand whether the participants believe that adoption of ASE tools and techniques are useful, both currently and in the future.
2. *RQ2: To what extent are developers in Thailand adopting software measurement, static code analysis, software testing, and continuous integration tools in their development?* This research question aims to understand the level of adoption of four types of ASE tools and techniques that are identified as challenges from the previous study (Ragkhitwetsagul et al. 2022) including software measurement, static code analysis, software testing. We also included continuous integration because it is closely related.
3. *RQ3: What are the challenges faced in the adoption of ASE tools and techniques?* By asking this question, we want to the identify the challenges that Thai software developers face when adopting the four types of ASE tools and techniques presented in RQ2. The findings can give us more understanding of the challenges Thai SSMEs faced when using such tools and techniques and how to tackle such challenges.
4. *RQ4: How do different groups of Thai developers compare?* We surveyed several groups of Thai developers which may differ on their knowledge and experiences of ASE tools and techniques. Thus, we compare four different groups of Thai developers to understand the similarities and differences in terms of ASE adoption.

The four research questions are investigated through a carefully designed survey questionnaire.

By creating the survey questionnaire based on the identified challenges from the previous study (Ragkhitwetsagul et al. 2022), the findings from this survey will complement the findings from that study and provide a more complete picture of the challenges faced by Thai SSMEs when adopting ASE tools and techniques.

2.2 Survey Creation

We created our survey's questionnaire by following the guidelines by Kitchenham and Pfleeger (2001-2003). We organized the questionnaire with the following structure:[1] (1) demographic information; (2) awareness and adoption of ASE tools; (3) adoption of different types of ASE tools: software measurements; static code analysis; continuous integration; software testing.

According to the principle of constructing a survey (Kitchenham and Pfleeger 2001-2003), we have searched for the relevant literature having a developer's survey or interviews of the four types of the ASE tools listed above. The identified literature is then used to create the survey questions, which are kept similar to allow comparison of the results. Regarding static analysis tools, we found that (Johnson et al. 2013) conducted interviews with 20 developers to investigate the low adoption of static analysis tools and Vassallo et al. (2018) surveyed 42 developers' perspectives on the usage of automatic static analysis tools and identified the contexts that the tools are used. Tomasdottir et al. (2020) surveyed 337 developers from the JavaScript community about ESLint adoption. Regarding software testing, we found two studies. First, Kochhar et al. (2015) surveyed Android and Windows app developers about their testing tools and the challenges they faced during automated testing of their apps.

[1] The questionnaire is available on our study website, https://muict-seru.github.io/ASETSI/publications.html

Second, Linares-Vasquez et al. (2017) surveyed 102 open-source contributors to Android projects about their practices when performing testing. Lastly, Hilton et al. (2016) surveyed 442 developers about why projects use or do not use continuous integration. We did not find any study with a survey or interview about software measurement tools. Thus, we have created a set of questions for this category similar to the questions in other categories.

We carefully created and tested the questionnaire. In places where the questionnaire asks about specific tool uses, it presents a list of known tools, partly extracted from previous work's questionnaires and extended by other typical tools for the category. In addition, an option to enter other tools as free text is provided. Some tools appear in multiple categories as they have multiple purposes. For example, SonarQube is a static code analysis tool, but it also provides software measurement and supports continuous integration and testing. Moreover, the four categories overlap. For example, some software measurements are acquired via static analysis (static software metrics) or during continuous integration and testing (dynamic software metrics). Furthermore, we performed a pilot study by having a graduate student (Ph.D.) who had software industry experience fill in the questionnaire in a read-aloud session with the first author. The pilot tester read the questions and explained their thinking regarding the questions and the answers. The first author noted down the comments and issues found in the questionnaire. After finishing the pilot study, we collected additional feedback about the overview and flow of the questionnaire from the pilot tester and improved our questionnaire accordingly. An initial distribution of the questionnaire to a limited number of participants revealed no issues and the questionnaire was used for the rest of the study. The structure of the questionnaire is shown in Table 1. The table also shows which research questions are addressed by each topic in the questionnaire.

The survey was fully anonymous and the questions on the questionnaire were bilingual (English and Thai). It was approved by both universities' research ethics boards.

2.3 Survey Distribution

The survey participants were recruited using non-probability convenient sampling (Wohlin et al. 2012). The inclusion criteria required participants with experience in software development within a company in Thailand. Participants were invited to take the survey via multiple channels, such as social media posts (Facebook), chat messages, emails, and online discussion boards/forums. All such channels were Thailand-specific and the call-for-participation was in Thai only. Aside from the general invitation, we also invited collaborating companies and organizations to participate in the survey. We also invited participants who joined local

Table 1 The structure of the online questionnaire and the research questions it addresses

Topic	No. of questions	RQ1	RQ2	RQ3	RQ4
Demographic	5				×
Awareness and adoption of ASE concepts and tools	4	×			×
Software measurements	10*		×	×	×
Static code analysis	14*		×	×	×
Software testing	15*		×	×	×
Continuous integration	7*		×	×	×

* This is the total number of questions. However, the actual number of questions shown to each participant depended on their answers to the preceding questions

workshops organized by the Thai researchers and participants of a software developer conference in Thailand to take the survey. The data collection was from May 2021 to November 2022.

2.4 Data Preparation and Analysis

The questionnaire was hosted on Google Forms and the answers were extracted when the survey was closed. The answers were then cleaned by removing participants who did not complete the questionnaire and participants who answered the questions randomly. The cleaned data were then analysed using descriptive statistics. The free text entered by the participants was Thai and such text was translated by the Thai researchers into English for analysis.

3 Results

After cleaning the survey data by removing ten participants who did not complete the questionnaire and one with answers that looked random, we were left with 103 participants. In the following, we will discuss the results of the survey for each of the parts.

3.1 Demographic Information

As the target of the survey was Thai software developers working in SSMEs, the demographic information was useful to check whether the target has been achieved. All free text entered was Thai which suggests that all participants were Thai. While we could not prevent participation from large software companies, the collected demographic information suggests that the risk of participants from large companies is low.

Table 2 shows the working experience of the participants. Overall, the working experience has a good distribution with 26 participants (25%) having more than 10 years of software development experience and 14 participants having less than a year of development experience (14%). Regarding the working experience of the participants at their current company, the largest portion (32 participants – 31%) work for their current company for less than a year. There are only 10 participants who work at the company for longer than 10 years. We also asked about their experience of working on the current project: 23 of the participants work on the current project for more than 2 years (22%), 11 between one and two years (11%) and the majority (69 participants – 67%) working on the project for up to one year.

The majority of the survey participants are software developers (69 participants – 68%), followed by managers or team leaders (34 – 33%). There are 9 participants with software quality assurance (QA) and tester roles (9%) and 7 designers (7%). Other roles include researcher (4), DevOps (2), tech support (1), technical officer (1), CEO (1), director (1),

Table 2 Working experience of the participants (in years)

	$<$ 1 year	1–2 years	3–5 years	6–10 years	$>$ 10 years
Overall	14 (14%)	21 (20%)	25 (24%)	17 (17%)	26 (25%)
Current company	32 (31%)	26 (25%)	22 (21%)	13 (13%)	10 (10%)

lecturer (1), and student (3). Most participants have multiple roles, but 48 (47%) have specified Developer as their only role and 19 (18%) have no roles other than Manager, Project Manager, or Team Leader. We also asked the size of the team that they are in (6 participants have not answered this question). Most participants (68 – 70%) work in small team sizes, mainly fewer than 10 people. The minimum team size is 1. The maximum team size is 180, which we considered an outlier. The median team size is 6.

3.2 Familiarity with Automated Software Engineering

The second part of the survey was about the participants' knowledge, experience, and opinions of Automated Software Engineering (ASE) tools and techniques. Asked *How familiar are you with the concept of automated software engineering?*, out of the 103 participants, only 13 answered they were not familiar with the concept of ASE, and 29 were slightly familiar. The rest were moderately (35), very (24), or extremely (2) familiar with the concept. Asked *To what extent are you adopting some of the automated software engineering tools in your work?*, 22 participants have not adopted ASE tools in their work, and the other 81 have adopted tools to varying extents (34 to some, 26 to a moderate, 17 to a good, and 4 to a great extent).

The 81 participants who have adopted ASE tools were asked whether it helps them to maintain the software more easily, deliver the software faster, deliver high-quality software, and whether the adoption of ASE tools is useful overall. They answered on the 5-level Likert scale of *not at all, slightly, moderately, very, or extremely*. All 103 participants were also asked about the future possible adoption of ASE tools. Table 3 shows the answers. In each row, we highlight the cells with different shades of grey according to their values. The high-value cells are highlighted using dark grey, while the low-value cells are highlighted using light grey. Only two participants did not find the current adoption of ASE tools useful and the majority (81%) found it to be very (34) or extremely (32) useful overall. The future adoption of ASE Tools is expected by the majority of the participants to be very or extremely helpful for maintaining software more easily (80%), delivering it faster (75%) and with high-quality (82%), and will be very or extremely useful overall (87%). From the highlighted cells, we can see that the current adoption of ASE Tools ranges from moderately to extremely (only 6% answered slightly or not at all). For future adoption, we can see from the shifts of the

Table 3 The current and future adoption of ASE tools

	N	S	M	V	E
The current adoption of ASE tools helps me to...					
... maintain the software more easily.	1	4	28	30	18
... deliver the software faster.	1	4	35	31	10
... deliver high-quality software.	1	4	24	34	18
is useful overall.	2	1	12	34	32
The future adoption of ASE tools will help me to...					
... maintain the software more easily.	0	2	19	52	30
... deliver the software faster.	0	3	23	52	25
... deliver high-quality software.	0	1	18	50	34
will become useful overall.	0	2	11	44	46

(N)ot at all, (S)lightly, (M)oderately, (V)ery, (E)xtremely grey-highlighted cells that the participants believe that ASE Tools adoption will be very helpful or extremely helpful (ranging from 75% to 82%).

To answer RQ1, most of the participants have some familiarity with ASE concepts. All participants believe that the future adoption of Automated Software Engineering tools will help them maintain the software more easily, deliver software faster, and deliver high-quality software. The majority of the participants have adopted some tools, but not all of them. Almost all participants who adopted tools agree that it helped them and that they are useful overall.

The results for RQ1 show that the experiences that were reported by our industrial partners in our previous investigation (Ragkhitwetsagul et al. 2022) are shared by Thai software practitioners in general (at least among the sample of 103 participants we surveyed) and the reported observations apply more generally.

3.3 Adoption of Automated Software Engineering

The largest part of the survey was about the adoption of specific ASE tools and techniques for software measurement, static code analysis, continuous integration, and automated testing. Each area had its own set of questions and, in the following, we will discuss each area separately.

3.3.1 Software Measurement Tools

We asked the participants about the software measurement tools they use. However, 23 were not familiar with such tools at all and only 50 have used them (49 out of the 53 who have not used such tools think that a team could benefit from using software measurement tools). Almost all participants who have used software measurement tools agree that they are useful (one participant did not answer the question) and only one participant of all participants who have used software measurement tools is not *currently* using such tools.

The following questions have only been given to the 49 participants who currently (at the time of the survey) use software measurement tools. 25 have used SonarQube, 3 have used Teamscale, 2 have used JDepend, 1 has used NDepend and 1 has used JHawk and found them useful. However, another participant has used JHawk and did not find it useful. 25 participants used other tools, e.g., Apache JMeter or Jacoco. However, some tools they mentioned are not primarily software measurement tools but provide some measurements, e.g., PyTest or ESLint. We also asked about the setup of the tools, shown in Table 4. About half of them use the tools in their continuous integration or code review process.

Table 4 Set up of the software measurement tools (multiple selections possible)

Do you use **software measurement tools** that run...	Answers
... continuously in your editor/IDE	17 (35%)
... in your editor/IDE when you tell them to run	20 (41%)
... as a part of pre-commit/pre-push checks	15 (31%)
... as a part of your build or CI process	23 (47%)
... as a part of your code review process	25 (51%)
... in a non-integrated/separate way	14 (26%)

We asked participants who have not used software measurement tools *"What prevented you from using software measurement tools?"* From the 36 responses, we found 3 main reasons. First, 23 participants stated that they neither have knowledge nor experience in using such tools. We list some of their free text answers below.

"I do not know about this type of tools and do not know their benefits."

"The projects are small and we do not know which tools to use. Normally, we use only the provided github/gitlab."

"I don't know this type of tools and don't know its benefits."

"I do not have time to study enough about this type of tools."

Second, 10 participants reported that they found software measurement tools not important or relevant for their projects such as

"Most of the work that I do cannot be measured by tools"

"I'm not sure if my team size is big enough for this kind of measurement."

"I don't think that it is important for now."

Lastly, one participant mentioned that he or she could not find an appropriate software measurement tool for their current software.

Overall, less than half of participants use software measurement tools but almost all of them find the tools useful. 25 out of 50 participants have used SonarQube. **The majority of the participants do not use software measurement tools but think they could be useful. The main reason for not using software measurement tools is the lack of knowledge and experience.** It seems the developers are not aware of tools and how to integrate them into their development workflow (i.e., lack of knowledge). Moreover, some of them do not find the tools useful, important or relevant for their work.

3.3.2 Static Code Analysis Tools

Out of the 103 participants, 37 have used static code analysis tools and found them useful to at least some extent. However, 3 out of 37 are currently not using static code analysis tools. Of the 66 who have not used such tools, 63 think that a team could benefit from using static code analysis tools. We asked the participants about the static code analysis tools they used. As this was a long list, Table 5 only shows tools with more than six participants who have used them (many of the tools that have only been used by a single participant have been found to be not useful by the participant). We observed that almost all the tools that have been used are free and/or open-source. Moreover, only a few participants found a tool not to be useful.

Some used tools have also appeared in the software measurement tools section as they provide measurements or metrics data. For example, SonarQube has been used by 17 participants to perform static analysis. It is not an inconsistency that SonarQube has been used by 25 participants for software measurement and by 17 participants for static code analysis as the questions included the purpose of the usage, e.g., *"Which of the following multi-language tools have you used to perform static code analysis?"*.

We again asked the participants about the setup of the tools (multiple selections). The results are shown in Table 6. The findings are similar to the study of Vassallo et al. (2018) in

Table 5 Usefulness of source code analysis tool usage

Tool	Not	Somewhat	Very
IntelliJ	3	8	12
ESLint		1	18
Eclipse		7	11
SonarQube		6	11
PyCharm		7	8
TSLint		3	12
JSLint		5	9
Pylint	1	5	6
Standard JS	3	4	4
JSHint		4	5
Checkstyle	1	3	5

that developers use static code analysis tools in three different contexts: local programming, code review, and continuous integration.

Similar to the software measurement tools, we asked the participants *“What prevented you from using static analysis tools?”* From 41 responses, we identified the following main reasons. First, similarly to the previous section, 78% of the participants (32) mentioned a lack of knowledge or experience, which is the majority of all the responses. Some responses include

“I don’t know this technology”

“I don’t know the benefits and drawbacks clearly”

“I do not know the software and its usage”

Second, 12% of the participants (5) stated that they could not find appropriate tools or they found the tools not needed such as

“Most of the work is about checking manually whether it is correct [according to the spec].”

“I cannot find the tools that fit well with my current software development tools”

“Previous projects are small, so I do not have a chance to use the tools.”

“I do not have any needs for using the tools with my current project”

Table 6 Set up of the static code analysis tools (multiple selections possible)

Do you use **static code analysis tools** that run...	Answers
... continuously in your editor/IDE	10
... in your editor/IDE when you tell them to run	15
... as a part of pre-commit/pre-push checks	15
... as a part of your build or CI process	16
... as a part of your code review process	10
... in a non-integrated/separate way	4

Table 7 Reasons to use continuous integration (CI), answered by 64 participants

Reason	Amount	Percent
CI helps us deploy more often	47	73.44%
CI makes integration easier	45	70.31%
CI provides a common build environment	38	59.38%
CI allows faster iterations	34	53.13%
CI can enforce a specific workflow	34	53.13%
CI helps us catch errors earlier	30	46.88%
CI allows testing across multiple platforms	26	40.63%
CI allows running tests on more powerful hardware	24	37.50%
CI lets us spend less time debugging	22	34.38%
CI helps us fix breaking builds sooner	20	31.25%

Lastly, 5% of the participants explained that they did not use static analysis tools because of the direction of the company or the management of the team. One participant said *"There is no agreement/planning at the beginning about using the tools."* Another participant mentioned that it depends on the *"Direction of the company"*.

Overall, only a minority of participants have used static code analysis tools and all of them found the tools to be useful. **The majority of the participants do not use static code analysis tools but think they could be useful. The main reason for not using static code analysis tools is the lack of knowledge and experience.** Almost all the tools that have been found useful are free and/or open-source.

3.3.3 Continuous Integration (CI) Tools

Continuous Integration (CI) is at the core of automated software development processes, and therefore we also asked the participants about their familiarity with CI tools. Of the 103 participants, 50 currently use CI, and 16 have used CI in the past. Of the 37 participants who have never used CI, 8 stated that they do not use CI because they are not familiar with it and 10 stated that they do not use CI because they do not have enough automated tests.

The reasons to use CI for the 66 participants who use or have used CI are shown in Table 7 (two have not answered the question). The three main reasons include helping the developers to deploy more often, making integration easier, and providing a common build environment. On the other hand, Table 8 shows the reasons for not using CI for the 37

Table 8 Reasons for not using continuous integration (CI), answered by 36 participants

Reason	Amount	Percent
The developers on my project(s) are not familiar with CI	9	25.00%
My project(s) do not have enough automated tests	9	25.00%
Maintenance costs are too high for CI systems	3	8.33%
Automating builds for my project(s) is not worth it	2	5.56%
CI does not bring value because we already do enough testing	1	2.78%
Setup costs are too high for CI systems	1	2.78%
Others	5	13.89%

participants who have never used CI (one participant has not answered the question). 22 participants do not currently use CI but they would like to use it in the future. Some of the reasons that make the developers not use CI include the developers having low familiarity with CI (9 responses), the projects do not have enough automated tests, thus cannot receive full benefits from CI (9 responses), and the CI maintenance costs are too high (3 responses).

The participants were also asked about the problems they encountered when trying to use CI. They encountered a range of problems shown in Table 9, with various degrees of frequency. Again, we highlight the cells according to their values using the same color scheme as before. There seems to be no clear pattern in the responses as the distribution of the degree of problems is similar. The problem of overly long build times has been encountered more often than the other problems. The majority of participants encountered the mentioned problems moderately frequently or very frequently. Regarding other problems, the majority of the participants faced them slightly or moderately often.

We have also asked the participants about the CI tools they used and Jenkins appears to be the one used most often (39 participants used it), followed by Azure DevOps Server (15 participants) and CircleCI (12 participants).

Overall, CI seems to be an accepted process. **The majority of participants use or have used CI.** However, the participants have encountered a range of problems when trying to use CI, with overly long build times being the most significant one.

3.3.4 Testing Tools

The last and largest part of the questionnaire was about testing as we found it to be a major challenge in our previous study (Ragkhitwetsagul et al. 2022). 100 (out of the 103) participants test their software in one way or another.

As shown in Fig. 1, 93 participants test manually and 54 participants (currently) use automated testing tools (47 participants do both). 62 participants have used automated testing tools. Regarding our question about type of testing they do (Fig. 2), 75 participants answered that they do unit testing, followed by functional testing (56 participants) and integration testing (54 participants). The least amount of testing is beta testing (18 participants) and regression testing (22 participants).

All the following questions have only been given to participants who currently use or have used automated testing tools (62 participants, however, one has not answered all questions). Table 10 shows for what purpose the participants use the tools. As expected, the tools are

Table 9 Problems encountered when trying to use CI

Problems encountered	N	S	M	V	E
Troubleshooting a CI build failure	6	16	19	14	9
Lack of support for the desired workflow	10	20	20	12	2
Maintaining a CI server or service	10	22	17	12	3
Lack of tool integration	9	23	17	11	2
Security and access controls	8	21	20	14	1
Automating the build process	11	18	17	13	3
Setting up a CI server or service	11	17	18	15	2
Overly long build times	5	14	18	21	5

(N)ot at all, (S)lightly, (M)oderately, (V)ery, (E)xtremely

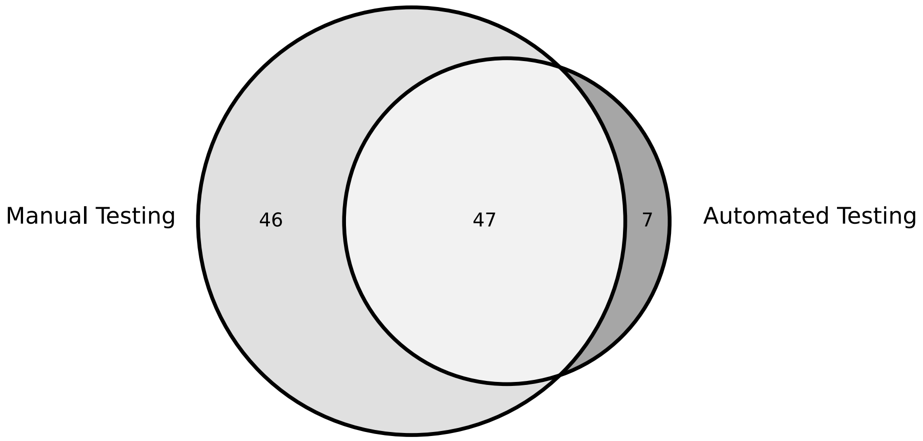


Fig. 1 How testing is performed by the participants

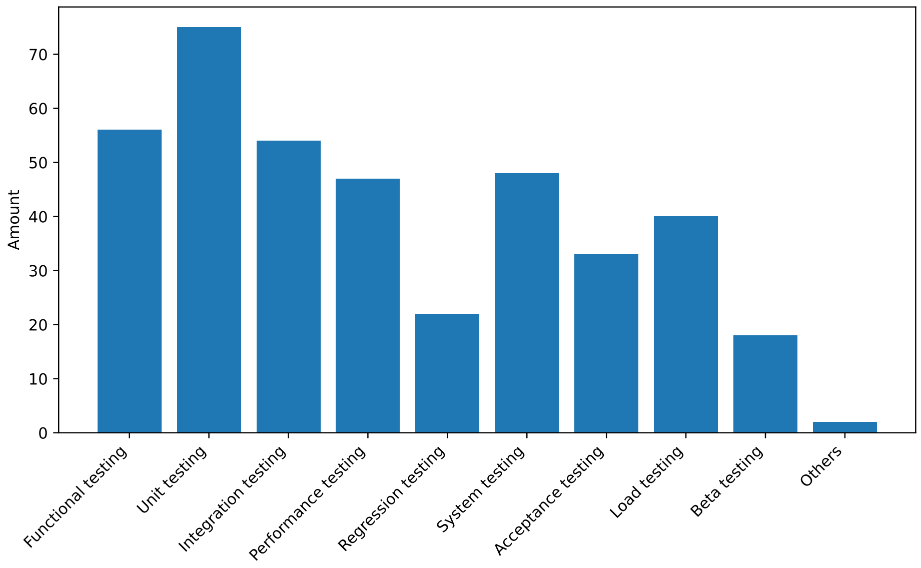


Fig. 2 Types of testing

Table 10 Usage of Testing Tools (ranked by the number of answers from 62 participants who have used testing tools)

Purpose		Purpose	
Executing test cases	55	Performing load testing	30
Creating test cases	48	Reporting bugs	23
Evaluating test results	43	Analysing code coverage	25
Finding potential bugs	38	Managing test suites	8

used most often for executing test cases (55 participants), followed by creating test cases (48 participants) and evaluating test execution results (43 participants). Postman (47 participants), Selenium (34 participants), Apache JMeter (26 participants), other (than JUnit) unit testing tools (21 participants), JUnit (20 participants), and Katalon Studio (15 participants) are the most often used tools, which possibly explains that automated testing is mainly performed on UI and API testing. We also asked the participants whether they use any tool for the automatic generation of test cases and only 21 participants said that they do. Automated testing tools like Selenium or Katalon Studio are used for the (manual) creation of test cases but are not able to automatically generate test cases.

We have asked the participants about the challenges they face using automated testing. As shown in Table 11, it can be observed that when discarding the participants who have no opinion on a testing challenge, for 6 out of the 9 testing challenges, the majority of the participants face serious or very serious challenges (highlighted with ∗). To measure the quality of the test cases, 45 participants used Statement Coverage, 24 participants used Instruction Coverage, 23 participants used Branch Coverage, and, surprisingly, 11 participants used Mutation Coverage (although no mutation testing tool had been mentioned by any participant). Two participants explicitly stated that they do not measure test case quality and six participants have left the answer empty. We asked the participants *"In your opinion, what are the top 2 things you need from an automated testing tool?"*.

Many participants (8 participants) have mentioned ease of use as one of the most important needs for automated testing tools, followed by good documentation and support (7 participants), required knowledge (5 participants), various other reasons (5 participants), accuracy and correctness (4 participants), speed (3 participants), and reduction of manual work (1 participants). There were 14 additional unrelated or unclear reasons that we could not group into any categories.

Overall, **the majority of participants have used automated testing tools. However, many participants face serious or very serious challenges when testing using automated testing. Ease of use and good documentation are the most important needs for automated testing tools.**

Although automated testing is a core element of continuous integration, not every adoption of continuous integration includes any form of automated testing. We have observed the same in our study. Of the 50 participants who are currently using continuous integration, 11 participants answered that they have never used automated testing tools and another two

Table 11 Testing Challenges. (N)o opinion, (D)o not face, (I)nsignificant, (S)erious, (V)ery serious

	N	D	I	S	V	
Time constraints	3	11	15	21	11	∗
Compatibility issues	3	12	12	29	5	∗
Lack of exposure to tools	2	6	18	28	6	∗
Unclear benefits of tools	4	26	14	12	4	
Poor documentation	4	11	21	19	5	
Steep learning curve	2	8	15	27	8	∗
Lack of support from supervisors/organisation	4	19	17	15	4	
Emphasis on development rather than testing	3	9	18	21	8	∗
Lack of software testing experience	2	10	14	26	7	∗

answered that they do not test their software. However, of the 11 participants who have never used automated testing tools, 8 do unit testing (although unit testing requires an automated unit testing framework).

3.3.5 Summary

The above discussion on the four different areas of ASE tools and techniques allows us to make the following general observation on the adoption of Automated Software Engineering tools and techniques in Thailand and answer our research questions about adoption and challenges.

To answer RQ2, Thai developers mostly do not use software measurement and static code analysis tools but think the tools can be useful. On the other hand, CI is well adopted. The majority of the participants use automated testing tools, although with several challenges.

Considering the challenges encountered, we have to distinguish challenges that prevented adoption from challenges that occurred during adoption and use. Regarding the challenges preventing adoption, it seems that a lack of knowledge and/or expertise is the main reason, at least for the adoption of software measurement tools, and static code analysis tools. The lack of experience, expertise and knowledge is a challenge during the adoption of automated software testing. The participants stated that they want automated testing tools that are easier to use and with good documentation. However, the challenges encountered during CI and automated testing are often technical.

To answer RQ3, the challenges faced in the adoption of ASE tools include a lack of knowledge, experience and exposure (software measurement, static code analysis, and automated testing tools); overly long build time (CI); time constraints, compatibility issues, and emphasis on development (automated testing tools).

We have seen that most of the participants have some familiarity with ASE concepts. However, what about the participants who have no familiarity? When analysing the data for the 13 participants, we can observe that 10 of them have limited work experience of up to two years. Despite having no familiarity with ASE concepts, five of the 13 participants answered that they adopted ASE tools to some extent (the other eight participants answered not at all). However, except for a participant who adopted a software measurement tool, the other four participants answered that they have not adopted tools for software measurement, static analysis, continuous integration, or automated testing. In contrast, two of the participants who answered that they have not adopted ASE tools at all, answered that they adopted software measurement, static analysis and automated testing tools or continuous integration and automated testing tools.

3.4 Comparing Participation Groups

We recruited four different groups of participants to be able to compare groups with different backgrounds, experiences, and motivations. Instead of asking more detailed questions, we directly targeted four different groups via four different channels.

1. First, this study is part of a larger project about the adoption of ASE techniques and tools in Thai SSMEs. As part of the project, we collaborate with four Thai SSMEs. We have

invited the project collaborators to participate in this study, leading to 20 participants from this group, named ‘collaborator group’ in the following. This is our first group of participants.

2. Next, we invited five companies that were in the process of achieving ISO/IEC 29110 Basic Profile certification. ISO/IEC 29110 is a software engineering standard for Very Small Entities (VSEs) which are enterprises, organizations, departments or projects having up to 25 people (Laporte and O’Connor 2016). Three of the authors involved in this research provided consulting services to companies for the preparation and certification of ISO/IEC 29110. Consequently, we invited them to participate in this study before initiating the consultancy process. From this group, 22 participants answered the questionnaire fully. This group of participants is named ‘ISO certification group’ in the following.
3. The third group of participants consists of 22 attendees of a Thai software developer conference, called DevMountain Tech Festival Season 2,[2] who participated in the study. The first and second authors attended the conference and invited the attendees of the conference to participate in the survey. The group is named ‘developer festival group’ in the following.
4. Lastly, we reached out to all the developers in Thailand. This fourth group of participants were invited via social media channels (a personal Facebook post and posts in a practitioner’s Facebook groups), leading to 39 participants. As this group had no specific audience, it is named ‘open group’ in the following.

Not only have the four groups different profiles, but they also have different motivations to participate in the study. The collaborator group is motivated by the collaboration with the researchers and the participation in the overall project. The ISO certification group is motivated by the ISO certification process and the contact with the researchers. The developer festival group is motivated by learning, networking, sharing, and skill enhancement. The motivation of the participants in the open group is less clear beyond their interest in sharing their experience with academic research. We also created two more groups for comparison. The first group consists of the participants who currently have a software developer role. This group will have the most up-to-date experience with ASE tools and practices. The other group consists of participants who have three or more years of development experience, which includes participants who currently have no software development role. Such more senior participants are expected to have more experience and a broader horizon.

3.4.1 Demographic

The four groups have a different distribution of software development experience as shown in Table 12. The baseline group consists of all participants and 43 of them (42%) have more than 5 years of experience and 26 (25%) have more than 10 years of experience. Participants from the developer festival group have the most experience, 59% of them have more than 5 years of experience (36% even more than ten years), The participants from the open group have a smilarily high experience, with 48% of them having more than 5 years of experience (38% even more than ten years). The collaborator group have 45% of the participants with more than 5 years of experience. In contrast, the participants from the ISO certification group have the least experience, with 59% of them having up to two years of experience (only 10% have more than 5 years of experience).

[2] https://www.eventpop.me/e/13422

Table 12 The participant's software development experience based on groups (in years)

	< 1Y	1–2Y	3-5Y	6-10Y	> 10Y	Total
Baseline (all)	14 (14%)	21 (20%)	25 (24%)	17 (17%)	26 (25%)	103
Developers only	8 (12%)	17 (25%)	20 (29%)	10 (14%)	14 (20%)	69
Senior	–	–	25 (37%)	17 (25%)	26 (30%)	68
Collaborators	4 (20%)	5 (25%)	2 (10%)	7 (35%)	2 (10%)	20
ISO Certification	4 (18%)	9 (41%)	7 (32%)	1 (5%)	1 (5%)	22
Developer Festival	1 (5%)	0 (0%)	8 (36%)	5 (23%)	8 (36%)	22
Open	5 (13%)	7 (18%)	8 (21%)	4 (10%)	15 (38%)	39

The lower experience of the participants from the ISO certification groups also shows in the time the participants worked for the current company, with only 9% working for more than 5 years there (19% from the developer festival group, 30% from the collaborator group, and 28% from the open group). This is potentially because they are very small entities that are starting their software business.

As discussed in Section 3.1, not all participants currently have a software developer role. The group of 69 participants who have currently a software developer role will therefore be considered, too. In that group, only 24 out of 69 participants (34%) have more than 5 years of experience and 14 (20%) have more than 10 years of experience. This indicates that the participants in the group of current developers have less experience than the baseline, which could be explained by career changes where software developers have advanced to management roles.

Another group that will be considered consists of the participants who have three or more years of development experience (which includes participants that currently have no software development role). The group (called the 'senior' group in the following) consists of 68 participants. 44 of them (55%) have more than five years of experience.

The difference in experience will have impacted the answers to the questions for the other categories which will be discussed next.

3.4.2 Automated Software Engineering Tools

As expected from the demographic profile, the participants from the developer festival group have the highest familiarity with ASE tools and techniques (shown in Table 13). 86% of them have at least a moderate familiarity and 45% are very familiar with them. 62% of the participants from the open group have at least a moderate familiarity, followed by the ISO certification group (46%) and the collaborator group (40%). Compared to all participants where 59% have at least a moderate familiarity and 25% are very familiar with them, the senior group has 72% with at least a moderate familiarity and 32% are very familiar with them, and the developers-only group has 58% with at least a moderate familiarity and 20% are very familiar with them. Overall, the senior group, the festival group, and the open group have a higher familiarity with ASE tools and techniques than the baseline group of all participants.

Similar observations can be made for the adoption of ASE tools (also shown in Table 13). From the developer festival group, 68% of participants have adopted ASE tools to at least a moderate extent (41% to a good or great extent). Again, the next group is the open group participants (46%), followed by ISO certification group participants (36%), and the collaborator group participants (30%). While the baseline group of all participants has 46% of

Table 13 The participant's familiarity with ASE tools and techniques

	N	S	M	V	E	Total
How familiar are you with the concept of automated software engineering?						
(N)ot at all, (S)lightly, (M)oderately, (V)ery, (E)xtremely.						
Baseline (all)	13 (13%)	29 (28%)	35 (34%)	24 (23%)	2 (2%)	103
Developers only	10 (14%)	19 (28%)	26 (38%)	13 (19%)	1 (1%)	69
Senior	3 (4%)	16 (24%)	27 (40%)	20 (29%)	2 (3%)	68
Collaborators	2 (10%)	10 (50%)	5 (25%)	3 (15%)	0 (0%)	20
ISO Certification	6 (27%)	6 (27%)	7 (32%)	3 (14%)	0 (0%)	22
Developer Festival	1 (5%)	2 (9%)	9 (41%)	8 (36%)	2 (9%)	22
Open	4 (10%)	11 (28%)	14 (36%)	10 (26%)	0 (0%)	39
To what extent are you adopting some of the automated software engineering tools in your work?						
(N)ot at all, (S)ome, (M)oderately, (G)ood, Gr(E)at.						
Baseline (all)	22 (21%)	34 (33%)	26 (25%)	17 (17%)	4 (4%)	103
Developers only	16 (23%)	25 (36%)	16 (23%)	9 (13%)	3 (4%)	69
Senior	12 (18%)	21 (31%)	18 (26%)	13 (19%)	4 (6%)	68
Collaborators	6 (30%)	8 (40%)	4 (20%)	2 (10%)	0 (0%)	20
ISO Certification	8 (36%)	6 (27%)	6 (27%)	2 (9%)	0 (0%)	22
Developer Festival	2 (9%)	5 (23%)	6 (27%)	6 (27%)	3 (14%)	22
Open	6 (15%)	15 (38%)	10 (26%)	7 (18%)	1 (3%)	39

participants adopted ASE tools to at least a moderate extent, the senior group has 51% and the developers-only group has 41%. Again, the senior group, the festival group, and the open group have higher adoption of ASE tools and techniques than the baseline group of all participants.

The participants from the developer festival group not only have the highest adoption of ASE tools, but they also have the best experience (shown in Table 14): 70% of them find ASE tools very or extremely helpful to maintain software more easily, 60% to deliver the software faster, and 75% to deliver high-quality software. However, while 75% of them find ASE tools overall very or extremely useful, in other groups the percentage is even higher: 79% of the collaborator group participants, 85% of the open group participants, and 86% of the ISO certification group participants. The baseline group of all participants has 59% of participants find ASE tools very or extremely helpful to maintain software more easily, 51% to deliver the software faster, 64% to deliver high-quality software, and 81% find ASE tools overall very or extremely useful. The senior group has 59% of participants find ASE tools very or extremely helpful to maintain software more easily, 52% to deliver the software faster, 66% to deliver high-quality software, and 77% find ASE tools overall very or extremely useful. Of the developers-only group, 55% of participants find ASE tools very or extremely helpful to maintain software more easily, 53% to deliver the software faster, 62% to deliver high-quality software, and 81% find ASE tools overall very or extremely useful.

Overall, the participants from the developer festival group and the open group have the highest familiarity with ASE tools and techniques, the highest adoption of ASE tools, and the best experience with ASE tools. In contrast, the other two groups have the lowest familiarity,

Table 14 The participant's experience with ASE tools

	N	S	M	V	E	Total
The current adoption of ASE tools helps me to...						
(N)ot at all, (S)lightly, (M)oderately, (V)ery, (E)xtremely.						
... maintain the software more easily.						
Baseline (all)	1 (1%)	4 (5%)	28 (35%)	30 (37%)	18 (22%)	81
Developers only	0 (0%)	3 (6%)	21 (40%)	18 (34%)	11 (21%)	53
Senior	1 (2%)	4 (7%)	18 (32%)	17 (30%)	16 (29%)	56
Collaborators	0 (0%)	2 (14%)	5 (36%)	5 (36%)	2 (14%)	14
ISO Certification	0 (0%)	0 (0%)	8 (57%)	6 (43%)	0 (0%)	14
Developer Festival	1 (5%)	1 (5%)	4 (20%)	9 (45%)	5 (25%)	20
Open	0 (0%)	1 (3%)	11 (33%)	10 (30%)	11 (33%)	33
... deliver the software faster.						
Baseline (all)	1 (1%)	4 (5%)	35 (43%)	31 (38%)	10 (12%)	81
Developers only	1 (2%)	3 (6%)	21 (40%)	23 (43%)	5 (9%)	53
Senior	0 (0%)	4 (7%)	23 (41%)	21 (38%)	8 (14%)	56
Collaborators	0 (0%)	1 (7%)	8 (57%)	4 (29%)	1 (7%)	14
ISO Certification	1 (7%)	0 (0%)	8 (57%)	5 (36%)	0 (0%)	14
Developer Festival	0 (0%)	2 (10%)	6 (30%)	10 (50%)	2 (10%)	20
Open	0 (0%)	1 (3%)	13 (39%)	12 (36%)	7 (21%)	33
... deliver high-quality software.						
Baseline (all)	1 (1%)	4 (5%)	24 (30%)	34 (42%)	18 (22%)	81
Developers only	1 (2%)	2 (4%)	17 (32%)	23 (43%)	10 (19%)	53
Senior	0 (0%)	4 (7%)	15 (27%)	22 (39%)	15 (27%)	56
Collaborators	0 (0%)	1 (7%)	8 (57%)	4 (29%)	1 (7%)	14
ISO Certification	1 (7%)	0 (0%)	5 (36%)	7 (50%)	1 (7%)	14
Developer Festival	0 (0%)	2 (10%)	3 (15%)	11 (55%)	4 (20%)	20
Open	0 (0%)	1 (3%)	8 (24%)	12 (36%)	12 (36%)	33
... is useful overall.						
Baseline (all)	1 (1%)	2 (2%)	12 (15%)	34 (42%)	32 (40%)	81
Developers only	1 (2%)	1 (2%)	8 (15%)	23 (43%)	20 (38%)	53
Senior	2 (4%)	1 (2%)	10 (18%)	24 (43%)	19 (34%)	56
Collaborators	0 (0%)	0 (0%)	3 (21%)	8 (57%)	3 (21%)	14
ISO Certification	0 (0%)	0 (0%)	2 (14%)	5 (36%)	7 (50%)	14
Developer Festival	1 (5%)	1 (5%)	3 (15%)	9 (45%)	6 (30%)	20
Open	1 (3%)	0 (0%)	4 (12%)	12 (36%)	16 (48%)	33

the least adoption, and the lowest experience, but they still find the adopted ASE tools very or extremely useful. The senior group reports a higher familiarity and a higher adoption compared to the baseline and the developers-only group. Their experience is similar to the baseline, but they find the current adoption not as useful as the baseline.

Note that while the senior group and the festival group have a higher familiarity and a higher adoption compared to the baseline, the two groups have a lower rate of participants finding ASE tools very or extremely useful.

3.4.3 Software Measurement Tools

The difference in experience for the four groups also shows up in the software measurement category, as shown in Table 15. Only one of the participants from the developer festival group is not familiar with software measurement tools, and 82% are at least moderately familiar with 41% even very or extremely familiar. The participants of the other three groups have a much lower familiarity. 35% (collaborator group) / 36% (ISO certification group) / 33% (open group) are not familiar with such tools, and only 30% / 36% / 44% are at least moderately familiar with only 5% / 5% / 8% very or extremely familiar. While 48% of the baseline participants are at least moderately familiar, it is 57% for the senior group and 46% for the developers-only group.

This is also reflected in the number of participants who have used such tools. It is 77% for the developer festival group and 40% / 41% / 41% for the other three groups. For all participants it is 49%, for the senior group 57%, and for the developers-only group 46%. However, all participants who have used tools found them useful. As also shown in Table 15, 94% of the participants from the developer festival group are using software measurement tools to at least a moderate extent (13% / 44% / 44% for the other three groups), but only

Table 15 The participants' familiarity with software measurement tools

	N	S	M	V	E	Total
How familiar are you with automated tools used for measuring your software?						
(N)ot at all, (S)lightly, (M)oderately, (V)ery, (E)xtremely.						
Baseline (all)	23 (22%)	31 (30%)	35 (34%)	12 (12%)	2 (2%)	103
Developers only	15 (22%)	22 (32%)	20 (29%)	11 (16%)	1 (1%)	69
Senior	9 (13%)	20 (29%)	27 (40%)	10 (15%)	2 (3%)	68
Collaborators	7 (35%)	7 (35%)	5 (25%)	1 (5%)	0 (0%)	20
ISO Certification	6 (27%)	8 (36%)	7 (32%)	1 (5%)	0 (0%)	22
Developer Festival	1 (5%)	3 (14%)	9 (41%)	8 (36%)	1 (5%)	22
Open	9 (23%)	13 (33%)	14 (36%)	2 (5%)	1 (3%)	39
Currently, to what extent do you use software measurement tools during your activities?						
(N)ot at all, (S)ome, (M)oderately, (G)ood, Gr(E)at.						
	N	S	M	G	E	Total
Baseline (all)	1 (2%)	21 (42%)	17 (34%)	9 (18%)	2 (4%)	50
Developers only	0 (0%)	14 (44%)	11 (34%)	5 (16%)	2 (6%)	32
Senior	1 (3%)	16 (41%)	14 (36%)	7 (18%)	1 (3%)	39
Collaborators	1 (13%)	6 (75%)	1 (13%)	0 (0%)	0 (0%)	8
ISO Certification	0 (0%)	5 (56%)	2 (22%)	2 (22%)	0 (0%)	9
Developer Festival	0 (0%)	1 (6%)	12 (71%)	3 (18%)	1 (6%)	17
Open	0 (0%)	9 (56%)	2 (13%)	4 (25%)	1 (6%)	16

24% currently use them to a good or great extent, while the 31% from the open group and 22% from the ISO certification group use them to a good or great extent – but nobody from the collaborator group uses such tools to a good or great extent. 56% of the baseline group participants are using such tools to at least a moderate ex tent, and it is also 56% of both the senior group and the developers-only group participants. 22% of the baseline group and the developers-only group use them to a good or great extent, while it is 20% for the senior group participants. It is interesting to see that while the current use of software measurement tools is very similar for the baseline, developers-only and senior groups, the same cannot be said for other groups of participants.

3.4.4 Static Code Analysis Tools

Static code analysis tools have been used by 68% of the participants in the developer festival group, 38% in the open group, 30% in the collaborator group, but only 5% (a single participant) in the ISO certification group. For the baseline group, it is 36% of the participants, for the senior group 46%, and the developers-only group 36%. Similar to above, it is again the senior group, the festival group, and the open group that have a higher past or current usage of static code analysis tools than the baseline group. Almost all found the usage at least moderately useful (a single participant from the open group found the usage somewhat useful who is also in the senior and the developers-only group). As shown in Table 16 and only for the participants who have used static code analysis tools, 80% of the participants from the developer festival group currently use such tools to at least a moderate extent and 47% of them to a good or great extent. While 83% of the collaborator group use them at least to a moderate extent, only a single participant (17%) uses them to a good or great extent. For the participants from the open group, it is 60% to at least a moderate extent and 33% to a good or great extent. While 73% of all participants currently use such tools to at least a moderate extent, it is 56% of the developers-only group and 71% of the senior group.

3.4.5 Continuous Integration (CI) Tools

As shown in Table 17, 73% of the participants of the developer festival group currently use CI and only 14% have never used CI. The other three groups have, again, much less experience in CI. Only 40% / 45% / 41% currently use CI, but 40% / 41% / 44% have never used CI.

Table 16 The participants' familiarity with static code analysis tools

	N	S	M	G	E	Total
Currently, to what extent do you use static analysis tools during your activities?						
(N)ot at all, (S)ome, (M)oderately, (G)ood, Gr(E)at.						
Baseline (all)	3 (8%)	7 (19%)	13 (35%)	9 (24%)	5 (14%)	37
Developers only	2 (8%)	5 (20%)	7 (28%)	7 (28%)	4 (16%)	32
Senior	2 (6%)	7 (23%)	8 (26%)	9 (29%)	5 (16%)	31
Collaborators	1 (17%)	0 (0%)	4 (67%)	1 (17%)	0 (0%)	6
ISO Certification	0 (0%)	0 (0%)	0 (0%)	0 (0%)	1 (100%)	1
Developer Festival	0 (0%)	3 (20%)	5 (33%)	7 (47%)	0 (0%)	15
Open	2 (13%)	4 (27%)	4 (27%)	2 (13%)	3 (20%)	15

Table 17 The participants' familiarity with continuous integration tools

	Y	P	N	Total
Do you currently use CI?				
(Y)es; No, but used in the (P)ast; No, and (N)ever used CI.				
Baseline (all)	50 (49%)	16 (16%)	37 (36%)	103
Developers only	36 (52%)	12 (17%)	21 (30%)	69
Senior	40 (59%)	11 (16%)	17 (25%)	68
Collaborators	8 (40%)	4 (20%)	8 (40%)	20
ISO Certification	10 (45%)	3 (14%)	9 (41%)	22
Developer Festival	16 (73%)	3 (14%)	3 (14%)	22
Open	16 (41%)	6 (15%)	17 (44%)	39

For the baseline group, it is 49% currently using CI and 36% never used CI. Of the senior group, 59% currently use CI and 25% never used CI, and of the developers-only group, 52% currently use CI and 30% never used CI. Overall, the senior group, the developers-only group, and the festival group have a higher experience than the baseline group. However, the difference in experience does not lead to a pattern in the problems encountered when using CI. For each of the problems and each group, the participants of the group can encounter the problem more frequently or less frequently. As the absolute number of participants in each group is small, we refrained from further statistical analysis to expose a pattern. It seems that more experience does not lead to fewer problems during the use of CI.

3.4.6 Testing Tools

The difference in experience also shows in the usage of automated testing tools where 90% of the participants in the developer festival have used such tools (shown in Table 18). In the other three groups, it is 53% / 48% / 59% of the participants. For the baseline group, it is 62% of the participants, for the senior group 67%, and for the developers-only group 60%. Only the senior group and the festival group have a higher experience than the baseline group. Regarding manual and automated testing, in the baseline group of all participants, 92% test manually and 54% with automated testing tools. Compared to the baseline group, the developers-only group (94%), the ISO certification group (100%), and the open group

Table 18 The participants' familiarity with testing tools

	M	A	E
How do you test your software?			
(M)anually; Use (A)utomated testing tools.			
Have you (E)ver used automated testing tools?			
Baseline (all)	94/102 (92%)	55/102 (54%)	62/100 (62%)
Developers only	65/69 (94%)	37/69 (54%)	41/68 (60%)
Senior	60/67 (90%)	40/67 (60%)	45/67 (67%)
Collaborators	18/20 (90%)	1/20 (5%)	10/19 (53%)
ISO Certification	22/22 (100%)	11/22 (50%)	10/21 (48%)
Developer Festival	16/21 (76%)	17/21 (81%)	19/21 (90%)
Open	38/39 (97%)	26/39 (67%)	23/39 (59%)

(76%) have a higher percentage of participants who test manually. Compared to the baseline group, the senior group (60%), the festival group (81%), and the open group (67%) have a higher percentage of participants who use automated testing tools.[3]

Regarding the different problems encountered during manual testing, the participants of the ISO certification group report more often that they do not face the problem or the problem is not significant than the participants in the other three groups. For four out of the five different problems, it is the developer festival group for which the participants report most often that the problem is serious or very serious (for the other problem, it is the collaborator group).

A similar pattern can be observed for the problems encountered during testing using automated testing tools. For seven out of nine problems, it is again the ISO certification for which the participants report most often that they do not face the problem or that the problem is insignificant (for the other two problems, it is the collaborator group). Also, for six out of nine problems, it is again the developer festival group for which the participants report most often that the problem is serious or very serious (for the other three problems, it is the collaborator group).

3.4.7 Summary

Although our observations on the four different participant groups cannot be generalized as the group sizes are not large enough, they show some interesting patterns. Most importantly, our comparison demonstrates how targeting different audiences lead to different demographics and different results in questionnaires. The developer festival group and the senior group have participants with more experience than the other groups and this affects the observations in the different categories. The participants from the two groups report a higher familiarity with ASE tools and techniques, a higher adoption of ASE tools, a higher familiarity with software measurement tools, have used software measurement or static code analysis tools more often, use CI tools more often, and have used automated testing tools more often. However, participants from the same group do not face fewer problems during CI and face more serious problems in testing. It seems that more experience does not lead to fewer or less serious problems.

To answer RQ4, different results can be observed for different participation groups of Thai developers as they have different demographics. The different results can be mainly observed in terms of experience with ASE tools and techniques. However, groups with more experience do not necessarily face fewer or less serious problems.

The notable difference in the results for the senior and the developer festival groups compared to the baseline group of all participants is that both groups have a longer software development experience (the senior group by construction and the developer festival group as demonstrated by the demographics – it is almost a subset of the senior group). Moreover, developers attending an event like the developer festival are likely more interested in advancing their knowledge and skills, which may explain why they are more familiar with ASE tools and techniques.

[3] There may have been a misunderstanding of the question because in the open group 26 participants reported that they use automated testing tools, but only 23 reported that they have ever used automated testing tools.

4 Comparison to Other Studies

This goal and the methodology of our study is built upon the previous studies. Thus, in this section, we compare our results to the findings of the previous studies for which we incorporated their survey questions in our questionnaire. Moreover, we compare our findings to related studies on large and ultra-large companies and to the findings from our previous study that motivated this study. Other related work is discussed in Section 6.

4.1 Comparisons to Previous Studies about Static Code Analysis, Testing, and CI

Our survey was designed so that the results can be compared to previous studies. In this section, we compare our results to the results of other studies that we included the questions from their studies in our questionnaire (Johnson et al. (2013); Vassallo et al. (2018); Tomasdottir et al. (2020); Kochhar et al. (2015); Linares-Vasquez et al. (2017); Hilton et al. (2016)). As discussed above, we did not find any previous questionnaire-based study on the adoption of software measurement. So, in the following subsection, we focus on static code analysis, CI, and testing.

4.1.1 Static Code Analysis

Compared to the study by Johnson et al. (2013), our findings are different. Johnson reported that the factors affecting the use or under-use of static analysis tools include (1) too many results or false positives in the tool's output, (2) weak support for collaboration, (3) difficulties in configuration, and (4) low result understandability. We mainly found that the tools are not adopted or underused due to the lack of knowledge or understanding of the developers. Nonetheless, the participants in Johnson's study are from large software companies, which may have more readiness and experience in adopting the static analysis tools than our participants in Thailand.

Consider the study of static analysis tools by Vassallo et al. (2018), our findings are also different. Vassallo et al. (2018) reported that 38% and 31% of their survey participants use automatic static analysis tools multiple times per day and on average once per day respectively. In our study, we found a lower percentage of 33% of the participants that are currently using static analysis tools. In terms of the most used tools, Vassallo et al. (2018) found FindBugs as the highest adopted tool (19%), followed by Checkstyle (18%). We found that IntelliJ is the most used static analysis tool (62%), followed by ESLint (51%). Among the top ten tools reported by their study and ours, the common tools, sorted by their popularity in our study, include ESLint (51%), SonarQube (46%), Pylint (32%), JSHint (24%), and Checkstyle (24%). These differences can also be potentially due to the different development context and the samples of the participants. Regarding the development activities that static code analysis tools are integrated, we found that local programming is the most popular activity either by running the tools continuously in IDE or running the tools in an ad-hoc manner (68%), while Vassallo et al. (2018) found that CI is the most popular activity (37%). Moreover, the differences can also be partially explianed by the different time of Vassallo et al. (2018)'s study (2018) and our study (2022), which affect the popularity of languages and development frameworks.

We incorporated the list of JavaScript linters from the study by Tomasdottir et al. (2020) in our survey questions regarding static analysis tools. From our survey, we found that ESLint is the most widely used JavaScript linter (19 out of 37 participants (51%) who use static code analysis tools use it), followed by TSLint (41%), JSLint (38%), Standard JS (30%), and JSHint (24%). Since the Tomasdottir et al. (2020)'s survey results did not report the number of linter adoption, we analysed their provided raw responses to compare with our findings. We found that from 337 survey responses, the majority of the participants similarly use ESLint (138 out of 337 – 80%). The ranking of other tools are slightly different. The second tool in their study is JSLint (41%), followed by TSLint (24%), JSCS (13%), Standard JS (10%), and other tools, such as Prettier (4%). We found a higher rate of TSLint adoption in our study although TSLint was deprecated February 2019. We did not ask the participants on the preset (i.e., default configurations) and the rules that the participants use in their linters as in their study.

4.1.2 Testing

Compared to the Kochhar et al. (2015)'s study of test automation by app developers at Microsoft, we found some different and similar results. They found that functional testing is the most popular type of testing compared to unit testing in our study. Beta testing is reported as the type of testing with the least popularity, which is similar to our study's findings. Microsoft's developers adopted more regression testing than Thai developers. The amount of regression testing is ranked the 5th compared to the 8th in our study. According to the study by Linares-Vasquez et al. (2017), they found that the most used tool for automated testing in Android applications is JUnit, followed by Roboelectric, and Robotium. In our study, we found that Postman is the most used tool for automated testing followed by Selenium, and Apache JMeter. This difference is possibly due to the wider range of participants in our study compared to only Android developers in their study.

4.1.3 Continuous Integration

In contrast, continuous integration appears to be an accepted practice in the software industry. However, the participants have encountered a range of problems when trying to use CI, with overly long build times being the most significant problem. Hilton et al. (2016) surveyed 442 GitHub developers and found that the two main reasons for the developers using CI are to avoid breaking builds (87.71%), and to catch bugs earlier (79.61%). The two main reasons of using CI for the developers in our study are to deploy more often (73.44%), and to make integration easier (70.31%). Regarding the reasons of not using CI, Hilton et al. (2016) found that developers do not use CI mainly because they are not familiar with CI (47.00%), followed by the projects not having enough automated tests (44.12%), which are very similar to the findings in our study.

4.2 Comparison to Related Studies from Large and Ultra-Large Companies

Besides comparing our study to the previous studies that we based our questionnaire on, we also compare our findings to other related studies on large and ultra-large companies.

Regarding software measurement tools, Morgenthaler et al. (2012) reported how Google measured and handled technical debt, specifically Build Debt, in their large code base. They showed that Google engineers were facing with several kinds of build debts including dependency debt, visibility debt, zombie targets, and dead flags. Compared to our study, some of the participants use software measurement tools that can measure and report technical debts such as SonarQube (25 participants). However, we did not ask their understanding of the concept of technical debts nor the types of technical debts that they are facing.

Regarding static analysis tools, Ayewah et al. (2007, 2008) report the experiences from adopting FindBugs, a static analysis tool, at Google. They found that Google has systematic effort in addressing defects reported by FindBugs. It is worth noting that the study has been performed in 2007, approximately 16 years before our study. Thus, Google has been using static analysis tools for at least more than a decade. Johnson et al. (2013) study the reasons for the low adoption of static analysis tools in practice by interviewing 20 software practitioners, of which 16 are professional software developers at a large company. They found that the false positives reported by the tools and the developer overload are key factors for such low adoption and suggest improvements by enhancing collaboration, developer's workflow integration, and easy configurations and defect explanations. The results of these studies show that developers from large companies already knew static analysis tools and they faced issues after using the tools (such as false positives). In contrast, our study shows that developers from SSMEs in Thailand are not familiar with static analysis tools, or not finding the tools useful. As a result, they do not use the tools.

Regarding software testing, Petrovic et al. (2022) and Petrović and Ivanković (2018) report the adoption of mutation testing and how to scale the technique to very large code base at Google. They presents a scalable and practical approach to mutation testing and evaluated the approach in a code-review-based setting involving more than 24,000 developers and more than 1,000 projects at Google (Petrovic et al. 2022). Another study by Petrovic et al. (2018) is performed on 1.9 million change set involving more than 30,000 developers at Google. They found that a special type of mutants called productive mutants are perceived as useful by developers. These studies shows that mutation testing is widely adopted at Google. Compared to our study, we found that only 11 participants out of 103 surveyed participants (approximately 11%) mentioned using mutation coverage, but without any mutation testing tool mentioned. We are unsure if the participants actually know the meaning of the term mutation coverage because it can also refer to the usage of mutation observation or tracking in tools such as Postman, or Selenium.

As previously discussed in the previous section (Section 4.1.2), the study of test automation at Microsoft by Kochhar et al. (2015) shows that functional testing is the most popular testing type compared to unit testing in our study while beta testing is the least popular testing type similar to our study. Moreover, Microsoft developers perofrm more regresstion testing than Thai SSMEs developers.

Regarding CI, Ståhl and Bosch (2013) conducted 22 semi-structured interviews with developers, testers, project managers and line managers of four projects at Ericsson AB about their experiences of continuous integration concerning whether CI is supporting agile testing, improving communication, increasing developer productivity, and improving project predictability. They found that the collected data supports, at least partly, all four areas. However, similar to our study, they found that the differences in experienced effects are large. There are several studies from Google, an ultra-large company, that report their adoption of CI and software testing tools and techniques. Memon et al. (2017) report that Google's Test Automation Platform (TAP), a CI system, integrated and tested on average more than 13,000 projects, 800,000 builds, and 150 million test runs per day. Due to this large scale execution, the TAP system was facing an issue of delays caused by out of memory errors, machine failures, and other infrastructure problems. Compared to our study, not all participants use

CI (50 currently use and 16 have used CI in the past). Among the participants who use or have used CI, we similarly found that the main issue they faced is overly long build times. However, the reason for the long build times may differ from Google because the participants also reported the issues of setting up a CI server or service and automating the build process. Thus, the overly long build times faced by developers in Thailand SSMEs may be caused by the lack of knowledge and experience in setting up and automating the CI build process instead of the scale of the build process as reported by Google. A study by Martensson et al. (2017); Mårtensson and Ståhl (2017) on two potential large companies (having more than 15,000 employees) identifies the main factors that could enable more frequent continuous integration including activity planning and execution, system thinking, speed, and confidence throuh test activities. Compared to our study, we found similar result that the speed of the CI process is one of the key challenges in adopting CI in SSMEs.

4.3 Comparison to Our Previous Study

Our previous study (Ragkhitwetsagul et al. 2022) shows that the four interviewed Thai SSMEs faced challenges of lack of code analysis and measurements and having low code quality. We similarly found that the majority of the participants in this study do not use software measurements and static code analysis tools. The reasons for not using software measurement tools include not being aware of such tools, finding software measurement tools not important or relevant for their projects, and not having an appropriate software measurement tool for their current software. Similarly, the reasons for not using static code analysis tools include not knowing such tools, not finding appropriate tools, not seeing the benefits of the tools, and the direction of the company or the management of the team.

Regarding software testing, we found both similarities and differences to the previous study. About half of the survey participants reported that they currently use automated testing, which is different from the finding in the previous study. This is possibly due to the large sample size including more experienced developers (as previously discussed in Section 3.4.6). Nonetheless, 46 participants reported that they only performed manual testing. We also find in this study that the majority of the participants perform unit testing, which differs from the finding in the previous study. Asked about the challenges in software testing, the participants in this study reported that they faced serious issues of compatiblity issues, lack of exposure to tools, steep learning curve, and lack of software testing experience. This is similar to the findings in the previous study that we found the lack of testing knowledge and time and the low test quality as parts of the main challenges.

4.4 Summary

The comparison of our results with previous studies revealed more differences than similarities, which is not surprising given the different contexts of the studies. Moreover, the previous studies were conducted in different countries and at different times, which may also explain some of the differences. Most of the previous studies also did not report the sizes of the companies of the participants, or they studied open-source software developers. This may also cause some of the differences in the results. Most of the previous studies focussed on static code analysis and CI tools, where the landscape of tools and languages has changed over time.

5 Lessons Learned

Our study showed that Thai developers are familiar with the concept of Automated Software Engineering and that ASE tools help them maintain software more easily, deliver software faster, and deliver high-quality software. A general observation is that despite being familiar with the concept of ASE, the participants have reported a lack of knowledge about actual ASE tools and reported challenges in their adoption.

When looking at the categories of ASE tools and techniques we studied, the picture is different for each category. The majority of the participants have not used software measurement tools and/or static code analysis tools, despite thinking that they could be useful. On the other hand, around half of the participants use continuous integration or automated software testing.

Our previous study with four companies showed that they faced many challenges in testing. This study confirmed such challenges as only about half of the participants currently use automated testing tools and almost all rely on manual testing. One reason may be that the majority of the participants faced serious or very serious challenges when testing using automated testing.

We observed inconsistencies in the answers of the participants, which may suggest that some participants have a different understanding of what automated testing tools are. For example, only 20 participants stated that they used a unit testing framework, while 43 stated that they do unit testing. Moreover, participants answered that they do unit testing (usually requiring automated testing tools) although they stated that they did not use automated testing tools.

The observation that eleven participants answered that they use mutation coverage (although nobody uses any tool for mutation testing) can also be explained by limited knowledge and misunderstanding of terminology. Ten of the eleven participants use web-testing tools like Postman, Selenium or Katalon Studio which have the ability to observe or track state mutations (via the MutationObserver Web API or Postman's MutationTracker). It is probable that the participants don't know the meaning of the term mutation coverage and assumed it refers to the usage of mutation observation or tracking.

Across all ASE tool categories, we observed a general preference for free and/or open-source tools. The cost of commercial tools or services often appears to be prohibitive for small companies in Thailand.

The lack of experience, expertise, and knowledge is a challenge appearing across software measurement, static analysis, and automated testing. In particular, for automated testing tools, ease of use, good documentation and support, and having the required knowledge are the most important needs. As having experience, expertise, and knowledge, and good documentation and support are making it easier to use tools, we observe that ASE tools need to be easy to use, even when the users' expertise, experience and knowledge are low. This observation is supported by the large amount of CI adoption where current CI tools and services are easy to use and adopt. Another way to address the lack of experience, expertise, and knowledge is the provision of awareness sessions, tool demonstrations and training sessions. However, resources for training are limited in SSMEs. Given the importance of ASE techniques and tools, they should be part of any Computer Science education in the future.

For a subset of the participants, we know in which company they are working and how they are developing the software. This allowed us to observe inconsistencies in answers, for example, for the same company, some participants answered that they do not currently use continuous integration while others answered that they do. Such inconsistencies are often occurring in questionnaire-based studies and ours is no exception.

We have not done a full comparative study with large or ultra-large software companies. However, the adoption of ASE techniques and tools is well documented for ultra-large companies (Winters et al. 2020), and therefore we would expect dramatically different results if the study had been done on developers working at such companies. Nonetheless, we compared the results of our study to related studies from large and ultra-large companies based on the previous studies as discussed in the previous section (Section 4.2).

6 Related Work

We present the related work in three parts: (1) adoption of ASE tools and techniques, (2) challenges of adopting software engineering practices in emerging countries, and (3) industry-academia collaboration in software engineering. The discussion of related work complements the comparison to other studies in Section 4. Both sections contain different information and conclusions.

6.1 Adoption of ASE Tools and Techniques

Large and ultra-large software companies have been actively inventing and adopting ASE tools and techniques in their software development practices. Google has been using several traditional and state-of-the-art ASE tools and techniques as describe in their book “Software Engineering at Google” (Winters et al. 2020) such as internal code search system, modern build system, internal code review tool, static analysis, and CI/CD. Moreover, the company also has their own dedicate software engineering research team that has been actively working on various ASE topics such as code review, static analysis, and testing as shown in some of their recent research publications (Henderson et al. 2023; Bieber et al. 2023; Krupennikov et al. 2022; Wang et al. 2021). Similarly, Microsoft, Meta, Amazon, and Ericsson AB also have invested tremendously in their research and development on new ASE tools and techniques to stay competitive (Liu et al. 2023; Jin et al. 2023; Bairi et al. 2023; Tuli et al. 2023; Chen et al. 2022; Lee et al. 2022; Effendi et al. 2023; Sawant and Sengamedu 2023; Luo et al. 2023; Stahl et al. 2017; Mårtensson et al. 2019).

Vassallo et al. (2018) report that developers mainly use static analysis tools in three different contexts: local environment, code review, and continuous integration. Tomasdottir et al. (2020) studied the adoption of JavaScript Linters. They interviewed 15 developers and surveyed over 300 developers who use JavaScript linters and identified benefits and challenges. The paper reports that a linter can help them maintain code consistency, save discussion time regarding which style to use, avoid ambiguous and complex code, and automate code reviews. However, they also found challenges in the adoption of the linters, including an agreement on the linter configuration, enforcing developers to follow the rules, and applying to existing projects. Bessey et al. (2010) also raised a concern about the understandability of outputs from automated static analysis tools. Especially, the study reports that over 30% of false-positive cases caused problems in real cases. Our study reveals that developers in Thailand have low adoption of static analysis tools due to a lack of awareness and knowledge. Nonetheless, when they adopt them, they use them in the same contexts as reported by Vassallo et al. (2018).

Wang et al. (2020) surveyed 151 software practitioners in 25 countries and identified challenges in automated testing adoption. They reflected that the low adoption is caused by the lack of guidelines on designing and executing automated testing. Karlsson et al. (2021) applied model-based automated testing of mobile applications in industry cases and reported their lessons learned. The paper reports that the knowledge transfer pipeline (e.g. finding a champion who is interested in automated tool adoption, using real examples) is an important factor in the success of the automated testing adoption. Maqbool et al. (2018) studied Pakistan's software companies with four years of experience in automated testing adoption. The study identified that the cost of buying automated testing tools and the effort of tool studying are the main hurdles stopping companies from using them. Kochhar et al. (2015) identified challenges in using automated testing tools, including project time constraints, configuration problems, and lack of experience and support. The study also found that some developers prefer to test their applications manually without using testing tools. The practitioner survey by Rafi et al. (2012) identified that the challenges in automation testing tool adoption are too high investments to buy tools, too complex project configurations, and lack of trustworthiness. Our study found a similar result that the benefits of adopting Automated Software Engineering tools, including automated software testing, may not outweigh the cost of adoption, especially in small-sized projects. Hilton et al. (2016) studied the reasons that software projects use or do not use continuous integration by surveying 442 developers. They found supporting reasons for using CI such as lowering the problems of breaking builds and catching bugs earlier. On the other hand, the top reasons for not using CI include developers not familiar with CI and no automated tests. Ståhl and Bosch (2013) performed a literature review and conducted semi-structured interviews on four projects at Ericsson AB to validate four hypotheses that CI is supporting agile testing, improving communication, increasing developer productivity, and improving project predictability. They could validate, at least partly, all four hypotheses. In a follow-up study (Ståhl and Bosch 2014), they performed a literature review and showed that there are differences in how the practice of continuous integration is interpreted and implemented.

6.2 Challenges of Adopting Software Engineering Practices in Emerging Countries

With a more general software engineering focus on emerging countries, previous work has highlighted the challenges such countries are facing when adopting software engineering practices, processes and methods. Mursu et al. (2000) identified risk factors to the Nigerian software industry and suggested that there are special requirements that should be taken into consideration in systems development methodology in Africa. Blake and Tucker (2006) explored software engineering and methodological development issues concerning three case studies from South Africa and presented a software engineering approach for these situations. Teka et al. (2016) focus on usability and user-centered design (UCD) in developing countries, in particular, in a private Ethiopian software development company. They report that most of the software companies in Ethiopia are young and inexperienced. Alamdy and Osman (2017) studied the software development industry in Sudan and identified key challenges as the insufficient description of software development procedures, communication methods and customer relations within companies. Based on experience in teaching software engineering in a developing country and in working within that country's local software industry, plus an overview of the literature on the African software industry, Osman (2012) proposes to improve the undergraduate software engineering curriculum in developing countries by supplementing it with the history, methods, techniques, anecdotes and experiences of the local software industries. Stefanovic et al. (2009) present the current level of the Serbian Information and Communication Technology (ICT) industry, define existing problems and suggest actions for improvement in the Serbian industry. They report that the Serbian ICT industry are predominantly small and medium privately-held companies. Khaksar and Khaksar (2015) examined the state of the software industry in Iran and suggested ways to improve its position. Kula et al. (2022) studied the challenges for software engineering in an emerging society and used Papua New Guinea as a case study.

6.3 Industry-Academia Collaboration (IAC) in Software Engineering

The presented study is part of a larger project about the adoption of ASE techniques and tools in Thai SSMEs which involves industry-academia collaboration (IAC). A literature review on the challenges of software process improvement within SSMEs was conducted by Balogun et al. (2022). They identified a list of 44 sustainability success factors that have a positive impact on implementing software process improvement efforts in SSMEs. Garousi et al. (2016) performed a systematic literature review on IAC in software engineering and their challenges and best practices. They identified 10 challenge themes and 17 best practice themes. However, they did not observe the size or location of the companies in their survey, and therefore it has not been included as a challenge or best practice. A follow-up study by Garousi et al. (2017) focused on the challenges and best practices of IAC in software testing. They observed differences between two groups consisting of projects in Canada and projects in Turkey. Of the ten projects they reported on, the majority of seven projects were from large companies (with more than 500 employees) and only one project was from a small company (with up to 100 employees). Daun et al. (2023) studied the success factors and barriers for technology transfer in software engineering. However, six of the seven companies involved in the study were large or ultra-large companies and only one company was an SME with about 200 employees, which was acknowledged as a threat to the generalizability of the study. Marijan and Sen (2022) reports on the experience of research collaboration and knowledge co-creation between industry and academia in software engineering. Their study involved eight organizations from which only one was an SME and four were large companies. Van de Vrande et al. (2009) investigated if open innovation practices are also applied by SMEs and focused on the motives and perceived challenges when SMEs adopt open innovation practices. In addition, they found significant differences between small-sized and medium-sized companies. Peças and Henriques (2006) identified best practices of collaboration between university and industrial SMEs based on the experience gained from more than 20 projects with Portuguese SMEs.

7 Threats to Validity

Internal validity: The selection of the survey participants is via convenient sampling and participants of workshops, which may affect the validity of the findings. We mitigated this threat by using inclusion criteria to control the eligibility of the participants. All entered free text was Thai and the demographic information suggests a low risk of participation from large companies. As discussed, almost all Thai software companies are SMEs. Moreover, from two of the four participant groups, we know that their companies are SSMEs (42 participants are from Collaborators and the ISO Certification groups). Moreover, some of the survey questions and the flow of the questions may cause confusion or misunderstandings as we found that many of the participants are not familiar with the concept of Automated Software Engineering. We partially mitigated this issue by having a pilot test of the survey before releasing it. We observed inconsistencies in answers which may affect the observations and conclusions. We observed large differences in responses between participants of the same or of different groups. This may be due to the different experience of the participants. However, it may also be due to different setups or tools used by the participants. For example, Ståhl and Bosch (2014) suggests that the kind of continuous integration that is used is affecting the experience. Lastly, there may be participants who are knowledgeable and adopt ASE tools and techniques on their own, even when the tools and techniques are not widely adopted at the company level. This may affect the findings of our study.

External validity: We received 103 responses from Thai software developers for this survey and the findings may not generalised to all software developers in Thailand. The scale of the survey prevented achieving a statistically representative sample of the Thai software development landscape. Almost 43% of the participants come from only nine companies and the diversity of the participants is limited. As we specifically targeted Thailand, the findings cannot be generalised to other countries. Lastly, 20 participants were from the four collaborating companies which may affect the generalizability of the findings. Nonetheless, the participants completed the survey questionnaire before the companies started collaborating with us on the adoption of ASE techniques and tools.

Ethics Considerations: The survey was anonymous and no personal information was collected. The participants were informed that the survey was conducted in Thai and that the results would be published in English. The survey was approved by the ethics committee of the authors' institutions. The ethics approval needed two steps: First, the survey was part of a larger project which was approved by both research ethics boards. As the survey questionnaire was created taking into account the first phase of the project, the final questionnaire was not known at the time of the first ethics approval. The approval of the final questionnaire was obtained in the second step.

As this study is part of a larger project, we are bound to strict regulations regarding ethics approval, data protection, and transfer of sensitive data out of the organisation that collected it and to a different country. While the survey was anonymous, it would still allow an informed insider to map some of the collected data to individuals in the cooperating companies. Therefore, we are only allowed to share and publish agglomerated data and were required to destroy the original data after the analysis and publication.

8 Conclusion

Much work has been done on the adoption of ASE tools and techniques in large and ultra-large software companies. However, little is known about the adoption of ASE tools and techniques in small and medium-sized software companies in emerging countries, and the challenges faced by such companies. Our study on the adoption of ASE tools and techniques showed that Thai software developers are somewhat familiar with ASE concepts and agree that adopting such tools would be beneficial. Moreover, the findings show that software measurement and static code analysis are not widely adopted by Thai developers while, on the other hand, continuous integration and automated testing are more widely adopted, although developers face several challenges. The challenges faced in the adoption of ASE tools include a lack of knowledge, experience and exposure (software measurement, static code analysis, and automated testing tools); overly long build time (CI); time constraints, compatibility issues, and emphasis on development (automated testing tools).

Having invited participants via different channels allowed us to observe differences in experience and adoption of ASE tools and techniques. Our observations suggest that more experience (and adoption) does not lead to fewer or less significant problems when using ASE tools. We, therefore, call for improvements in ASE tools, in particular in automated testing tools, to be easier to use to lower the barrier to adoption by software developers, in particular in small and medium-sized software companies with limited resources. However, such improvement will not address the lack of knowledge, experience and exposure of the developers. Such a lack can only be improved by training and education.

Acknowledgements This work is supported by the Newton Fund https://www.britishcouncil.org/education/he-science/newton-fund and funded by the Royal Academy of Engineering, UK (RAEng) https://www.raeng.org.uk/ and TSRI, Thailand https://www.tsri.or.th/ (Industry Academia Partnership Programme - 18/19 - Grant No. IAPP18-19/74). It was approved by both universities' research ethics boards.

Funding Open access funding provided by Mahidol University.

Data Availability The data associated with this study cannot be released due to the absence of ethical approval for its dissemination. The data collected for this research contain sensitive and potentially identifying information about the participants involved. In accordance with the ethical guidelines and regulations governing human subjects research, sharing the data without appropriate ethical clearance would compromise the privacy and confidentiality of the participants.

References

Alamdy S, Osman R (2017) Software industry practice in Africa: case study Sudan. In: 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), IEEE, vol 1, pp 743–748

Ayewah N, Pugh W, Morgenthaler JD, Penix J, Zhou Y (2007) Evaluating static analysis defect warnings on production software. In: Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, ACM, pp 1–8

Ayewah N, Pugh W, Hovemeyer D, Morgenthaler JD, Penix J (2008) Using Static Analysis to Find Bugs. IEEE Softw 25(5):22–29

Bairi R, Sonwane A, Kanade A, D C V, Iyer A, Parthasarathy S, Rajamani S, Ashok B, Shet S (2023) Codeplan: Repository-level coding using llms and planning. In: Neural Information Processing Systems Workshop on Foundation Models for Decision Making (FMDM-NeurIPS)

Balogun AO, Almomani M, Basri S, Almomani O, Capretz LF, Khan AA, Gilal AR, Baashar Y (2022) Towards the sustainability of small and medium software enterprises through the implementation of software process improvement: Empirical investigation. J Softw: Evol Process 34(8):e2466

Bessey A, Block K, Chelf B, Chou A, Fulton B, Hallem S, Henri-Gros C, Kamsky A, McPeak S, Engler D (2010) A few billion lines of code later: using static analysis to find bugs in the real world. Communications of the ACM 53(2)

Bieber D, Goel R, Zheng D, Larochelle H, Tarlow D (2023) Static prediction of runtime errors by learning to execute programs with external resource descriptions. In: Proceedings of the 11th International Conference on Learning Representations (ICLR '23)

Blake E, Tucker W (2006) Socially aware software engineering for the developing world. In: ST-Africa 2006 Conference Proceedings

Chen L, Abreu R, Akomolede T, Rigby PC, Chandra S, Nagappan N (2022) Leveraging test plan quality to improve code review efficacy. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), pp 1320-1330

Daun M, Brings J, Aluko Obe P, Tenbergen B (2023) An industry survey on approaches, success factors, and barriers for technology transfer in software engineering. Practice and Experience, Software

Digital Economy Promotion Agency (2021) Thailand's digital industry survey 2020 and 3-year prediction. https://www.depa.or.th/th/article-view/index-3-years-since-2563

Digital Economy Promotion Agency (DEPA), Thailand (2022) Results of the survey and assessment of digital industry 2022. https://www.depa.or.th/th/article-view/income-data-statistics-2565

Effendi SDB, Cirisci B, Mukherjee R, Nguyen H, Tripp O (2023) A language-agnostic framework for mining static analysis rules from code changes. In: Proceedings of 2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '23)

Commission European (2020) European Commission, User guide to the SME definition. Tech Rep 410. https://doi.org/10.2873/677467

Garousi V, Petersen K, Ozkan B (2016) Challenges and best practices in industry-academia collaborations in software engineering: A systematic literature review. Inf Softw Technol 79:106–127

Garousi V, Eskandar MM, Herkiloğlu K (2017) Industry-academia collaborations in software testing: experience and success stories from Canada and Turkey. Software Qual J 25(4):1091–1143

Henderson TAD, Dorward B, Nickell E, Johnston C, Kondareddy A (2023) Flake aware culprit finding. In: Proceedings of the 16th IEEE International Conference on Software Testing, Verification and Validation (ICST 2023), https://hackthology.com/flake-aware-culprit-finding.html

Hilton M, Tunnell T, Huang K, Marinov D, Dig D (2016) Usage, costs, and benefits of continuous integration in open-source projects. In: International Conference on Automated Software Engineering (ASE '16)

Jin M, Shahriar S, Tufano M, Shi X, Lu S, Sundaresan N, Svyatkovskiy A (2023) InferFix: End-to-End Program Repair with LLMs over Retrieval-Augmented Prompts. In: The ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23)

Johnson B, Song Y, Murphy-Hill E, Bowdidge R (2013) Why don't software developers use static analysis tools to find bugs? In: International Conference on Software Engineering (ICSE '13)

Karlsson S, Causevic A, Sundmark D, Larsson M (2021) Model-based automated testing of mobile applications: An industrial case study. In: Proceedings of the 14th IEEE International Conference on Software Testing, Verification and Validation Workshop (ICSTW '21)

Khaksar E, Khaksar A (2015) Study of the software industry problems in Iran. American J Syst Softw 3(1):20–23

Kitchenham B, Pfleeger SL (2001-2003) Principles of survey research (6 parts). ACM SIGSOFT Software Engineering Notes 26(6), 27(1), 27(2), 27(3), 27(5), 28(2)

Kochhar PS, Thung F, Nagappan N, Zimmermann T, Lo D (2015) Understanding the test automation culture of app developers. In: International Conference on Software Testing, Verification and Validation (ICST '15)

Krupennikov D, Hester W, Cortes D, Cole C (2022) Mobile Applications On-Device Testing at Google scale. Tech. rep, Google

Kula RG, Treude C, Hata H, Baltes S, Steinmacher I, Gerosa MA, Amini WK (2022) Challenges for inclusion in software engineering: The case of the emerging Papua New Guinean society. IEEE Softw 39(3):67–76

Laporte CY, O'Connor RV (2016) Systems and Software Engineering Standards for Very Small Entities: Accomplishments and Overview. Computer 49(8):84–87. https://doi.org/10.1109/MC.2016.242

Lee N, Abreu R, Nagappan N (2022) Code quality prediction under super extreme class imbalance. In: 2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW '22), pp 99–104

Linares-Vasquez M, Bernal-Cardenas C, Moran K, Poshyvanyk D (2017) How do developers test Android applications? In: International Conference on Software Maintenance and Evolution (ICSME '17)

Liu X, Jang J, Sundaresan N, Allamanis M, Svyatkovskiy A (2023) Adaptivepaste: Intelligent copy-paste in ide. In: The ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23)

Luo L, Mukherjee R, Tripp O, Schäf M, Zhou Q, Sanchez D (2023) Long-term static analysis rule quality monitoring using true negatives. In: Proceedings of 2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '23), pp 315–326

Maqbool B, Rehman FU, Abbas M, Rehman S (2018) Implementation of software testing practices in Pakistan's software industry. In: International Conference on Management Engineering, Software Engineering and Service Sciences (ICMSS '18)

Marijan D, Sen S (2022) Industry-academia research collaboration and knowledge co-creation: patterns and anti-patterns. ACM Transactions on Software Engineering and Methodology (TOSEM) 31(3):1–52

Martensson T, Stahl D, Bosch J (2017) Continuous Integration Impediments in Large-Scale Industry Projects. In: Proceedings of the 2017 IEEE International Conference on Software Architecture (ICSA '17), IEEE, November 2018, pp 169–178, 10.1109/ICSA.2017.11, http://ieeexplore.ieee.org/document/7930214/

Mårtensson T, Ståhl D (2017) Bosch J (2018) Enable more frequent integration of software in industry projects. J Syst Softw 142:223–236. https://doi.org/10.1016/j.jss.2018.05.002

Mårtensson T, Ståhl D, Bosch J (2019) Test activities in the continuous integration and delivery pipeline. Journal of Software: Evolution and Process 31(4):1–22. https://doi.org/10.1002/smr.2153

Memon A, Zebao Gao, Bao Nguyen, Dhanda S, Nickell E, Siemborski R, Micco J (2017) Taming Google-scale continuous testing. In: Proceedings of the IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP '17), IEEE, pp 233–242

Morgenthaler JD, Gridnev M, Sauciuc R, Bhansali S (2012) Searching for build debt: Experiences managing technical debt at Google. In: Proceedings of the Third International Workshop on Managing Technical Debt (MTD '12), IEEE, pp 1–6

Mursu A, Soriyan HA, Olufokunbi K, Korpela M (2000) Information systems development in a developing country: Theoretical analysis of special requirements in Nigeria and Africa. In: Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, IEEE, pp 10–pp

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

Ozkaya I (2022) The developer nation. IEEE Software 39(1)

Peças P, Henriques E (2006) Best practices of collaboration between university and industrial smes. Benchmarking: An International Journal 13(1/2):54–67

Petrović G, Ivanković M (2018) State of mutation testing at google. In: Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '18), ACM, pp 163–171

Petrovic G, Ivankovic M, Kurtz B, Ammann P, Just R (2018) An Industrial Application of Mutation Testing: Lessons, Challenges, and Research Directions. In: Proceedings of the IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW '18), IEEE, pp 47–53

Petrovic G, Ivankovic M, Fraser G, Just R (2022) Practical Mutation Testing at Scale: A view from Google. IEEE Trans Software Eng 48(10):3900–3912

Rafi DM, Moses KRK, Petersen K, Mäntylä MV (2012) Benefits and limitations of automated software testing. In: International Workshop on Automation of Software Test (AST '12)

Ragkhitwetsagul C, Krinke J, Choetkiertikul M, Sunetnanta T, Sarro F (2022) Identifying software engineering challenges in software SMEs: A case study in Thailand. In: International Conference on Software Analysis, Evolution and Reengineering (SANER '22)

Sawant N, Sengamedu SH (2023) Code compliance assessment as a learning problem. In: Proceedings of 2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '23), pp 445–454

Ståhl D, Bosch J (2013) Experienced benefits of continuous integration in industry software product development: A case study. In: Proceedings of the IASTED International Conference on Software Engineering, January, pp 736–743, https://doi.org/10.2316/P.2013.796-012

Ståhl D, Bosch J (2014) Modeling continuous integration practice differences in industry software development. J Syst Softw 87(1):48–59

Stahl D, Hallen K, Bosch J (2017) Continuous Integration and Delivery Traceability in Industry: Needs and Practices. In: Proceedings of the 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA'17), IEEE, October, pp 61–65, https://doi.org/10.1109/SEAA.2017.19, http://ieeexplore.ieee.org/document/8051329/

Stefanovic M, Matijevic M, Devedzic G (2009) Ict industry in Serbia: Condition and improvement by QMS. World Review of Science, Technology and Sustainable Development 6(2–4):259–277

Sunetnanta T, Suwannaroj S, Sangpar P (2016) ISO/IEC 29110 for Competitiveness – Challenges of Digital Cluster Development in Thailand. Tech. rep., ISO/IEC JTC 1/SC 7 Working Group 24

Teka D, Dittrich Y, Kifle M (2016) Usability challenges in an Ethiopian software development organization. In: Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering, pp 114–120

Tomasdottir KF, Aniche M, van Deursen A (2020) The adoption of JavaScript linters in practice: A case study on ESLint. IEEE Transactions on Software Engineering 46(8)

Tuli S, Bojarczuk K, Gucevska N, Harman M, Wang X, Wright G (2023) Simulation-driven automated end-to-end test and oracle inference. In: 2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '23), pp 122–133

Vassallo C, Panichella S, Palomba F, Proksch S, Zaidman A, Gall HC (2018) Context is king: The developer perspective on the usage of static analysis tools. In: International Conference on Software Analysis, Evolution and Reengineering (SANER '18)

Van de Vrande V, De Jong JP, Vanhaverbeke W, De Rochemont M (2009) Open innovation in smes: Trends, motives and management challenges. Technovation 29(6–7):423–437

Wang P, Bangert J, Kern C (2021) If It's Not Secure, It Should Not Compile: Preventing DOM-Based XSS in Large-Scale Web Development with API Hardening. In: Proceedings of the 43rd International Conference on Software Engineering (ICSE '21)

Wang Y, Mäntylä MV, Demeyer S, Wiklund K, Eldh S, Kairi T (2020) Software test automation maturity: A survey of the state of the practice. In: International Conference on Software Technologies (ICSOFT '20)

Winters T, Manshreck T, Wright H (2020) Software Engineering at Google, 1st edn. O'Reilly Media, Inc

Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in Software Engineering, vol 9783642290. Springer, Berlin Heidelberg, Berlin, Heidelberg

Yang J (2022) Building for the 99% developers. https://future.com/software-development-building-for-99-developers

Chaiyong Ragkhitwetsagul is a lecturer at the Faculty of Information and Communication Technology (ICT), Mahidol University, Thailand. He received a Ph.D. degree in Computer Science at University College London. He works collaboratively with small and medium-sized software companies in Thailand to help improving their software processes and products. His research interests include software dependencies, software visualization, code search, code clone detection, and mining software repositories. More info at https://cragkhit.github.io.

Jens Krinke is an Associate Professor in the Software Systems Engineering Group at the University College London (UCL), and Director of CREST, the Centre for Research on Evolution, Search, and Testing.

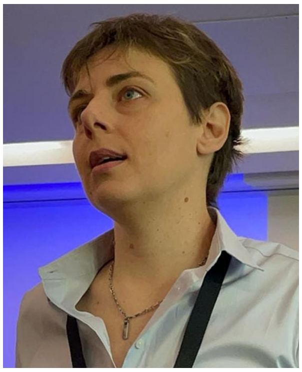
His main focus is empirical software engineering and software analysis for software engineering applications and purposes. He has over 20 years of experience in software reuse and software similarity research.

Before joining the University College London, he was at King's College London and the FernUniversitaet in Hagen, Germany, where he also worked on aspect mining and e-learning applications for distant teaching of software engineering.

He received his PhD in computer science from the University of Passau, Germany, and is well known for his work on program slicing and clone detection since then.

Morakot Choetkiertikul is an Assistant Professor at the Faculty of Information and Communication Technology (ICT) at Mahidol University, Thailand, where he co-founded the Software Engineering Research Unit (SERU). He earned his Ph.D. in Computer Science from the University of Wollongong (UOW), Australia. His research focuses on leveraging AI solutions to enhance software quality and software development processes.

Thanwadee Sunetnanta is an assistant professor at the Faculty of Information and Communication Technology (ICT), Mahidol University, Thailand. She received a Ph.D. in distributed software engineering from the Department of Computing, Imperial College, London. Her research interests include requirements engineering, software process improvement, qualitative approaches to software quality, secure software engineering, and green software.



Federica Sarro is a Professor of Software Engineering at University College London (UK), where she is the Head of the Software System Engineering group.

She has extensive academic and industrial expertise in Automated Software Engineering, with a focus on software management, requirements, optimisation, testing and repair.

On these topics she has published over 100 peer-reviewed scholarly articles, and she has been invited to give keynotes and present her work world-wide at several academic and industrial international events. Professor Sarro has obtained numerous awards and generous funding for her research. She has worked in collaboration with several companies including Bloomberg, Google, Meta, and Microsoft. In 2021, she was awarded the Rising Star Award by the IEEE Technical Community on Software Engineering in recognition of her "excellence in Software Engineering research with scholarly and real-world impact".

Authors and Affiliations

Chaiyong Ragkhitwetsagul[1] · **Jens Krinke**[2] · **Morakot Choetkiertikul**[1] · **Thanwadee Sunetnanta**[1] · **Federica Sarro**[2]

Chaiyong Ragkhitwetsagul
chaiyong.rag@mahido.ac.th
https://cragkhit.github.oi

Jens Krinke
j.krinke@ucl.ac.uk
http://www.cs.ucl.ac.uk/staff/j.krinke

Thanwadee Sunetnanta
thanwadee.sun@mahidol.ac.th
http://mucc.mahidol.ac.th/∼ittth

Federica Sarro
f.sarro@ucl.ac.uk
http://www.cs.ucl.ac.uk/staff/f.sarro

1 Faculty of Information and Communication Technology (ICT), Mahidol University, Nakhon Pathom, Thailand

2 UCL Computer Science, University College London, London, UK