Towards Understanding Industry's Perspectives on the Software Quality Characteristics: A Survey

Mert Ozkaya and Nurdan Canbaz
Department of Computer Engineering, Yeditepe University, Istanbul, Turkey

Keywords: Software Quality, Survey, ISO SQuaRE Quality Standard, Software Modeling Languages, Analysis Tools.

Abstract: The ISO SQuaRE software quality standard categorises software quality into eight different characteristics: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability and portability. Each quality characteristic is further considered in terms of a cohesive set of sub-characteristics. In this paper, a survey has been conducted with the goal of understanding which software quality characteristics are popular in industries and any modeling languages and tools used for analysing the software quality. The survey has been performed with 16 experienced practitioners, who work for a Turkish software company and have been requested to answer a pre-determined set of questions. The survey results lead to many interesting outcomes, some of which are as follows: *(i)* maintainability and performance efficiency are the top-popular characteristics, while portability is the least, *(ii)* time behaviour is the top-considered performance sub-characteristic, *(iii)* interoperability is the top-considered compatibility sub-characteristic, *(iv)* learnability and appropriateness recognisability are the top-considered usability sub-characteristics, *(v)* availability is the top-considered reliability sub-characteristic, *(vi)* confidentiality and integrity are the top-considered security sub-characteristics, *(vii)* modularity and reusability are the top-considered maintainability sub-characteristics, *(viii)* none of the participants use any modeling languages for analysing the software quality early on, and *(ix)* the participants use COTS tools for analysing the software implementation.

1 INTRODUCTION

Today, software systems have penetrated in almost all areas of human life and been actively used in various industries such as automative (Haghighatkhah et al., 2017), avionics (Marques and Cunha, 2013), military and defense (Çiflikli et al., 2012), telecommunications (Nitze and Schmietendorf, 2015), finance (Sajić et al., 2017), healthcare (Richardson et al., 2016), etc. Given also the ever-increasing advancements of technology and the customer demands, the expectations from software systems have been increasing rapidly. This essentially raised the issue of software quality, which is concerned with how well the software system meets its requirements. While many attempts have been made on explaining how the software quality should be understood (Kan, 2014; Kitchenham and Pfleeger, 1996; Tian, 2005; Cavano and McCall, 1978), one can basically consider it as the software system's compliance with the user requirements that can either be functional or non-functional. In the mid eighties, Garvin (Garvin, 1984) proposed a seminal approach that categorises the software quality into different views, which are transcendental, user, manufacturing, product, and value-based. The transcendental view considers the software quality in terms of its elegance and promotes the idea that one cannot measure the software quality but simply try their best to maximise the level of elegance. The user view is concerned with the user expectations and ensuring that the software system meets all the user requirements. The manufacturing view is concerned with the process followed in developing and deploying the software system and ensuring that the software system is developed right that conforms to the requirements and specifications of the system and thus any potential software faults are minimised. The product view is concerned with the internal and external qualities of a software system and how internal quality impacts on the external quality. The value-based view is concerned with the considerations of the software quality from multiple views discussed above and their consistencies with each other.

As indicated in (Ozkaya, 2018), many software modeling languages have been proposed for the modeling of software systems and their analysis for some quality properties. So, the quality issues may be detected early in the design phase before implement-

Ozkaya, M. and Canbaz, N.
Towards Understanding Industry's Perspectives on the Software Quality Characteristics: A Survey.
DOI: 10.5220/0007742004170426
In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2019)*, pages 417-426
ISBN: 978-989-758-375-9

ing the systems. While some languages support the exhaustive model checking or theorem proving techniques for proving the correctness software systems, some languages are supported with the analysis tools that analyse the software models for an quality properties of interest. AADL (Feiler et al., 2006) is, for instance, one of the most well-known software modeling languages for the embedded systems domain. AADL is supported by various analysis tools that allow for the analysis of the embedded software models for many properties, including latency, schedulability, timing, and resource utilisation. AADL's tools also generate C code from the embedded software models. Java Modeling Language (Chalin et al., 2005) is another highly popular software modeling language, which is tailored to Java and allows for combining the software modeling with implementation. Indeed, Java methods can be annotated with contract specifications and, using the exhaustive verifier tools available, the Java programs can be proved for the contractual specifications. Another well-known tool is UPPAAL (Larsen et al., 1997), which supports the modeling, simulation and analysis of the real-time systems. Using UPPAAL, one can specify their real-time systems model visually using timed automata, simulate its behaviour, and perform exhaustive model checking for any safety and liveness properties.

While software quality is one of the most crucial topics in software engineering and many languages (or tools) have been existing for promoting the quality software systems, it is not yet clearly known how practitioners in industries approach towards the software quality in their software development projects. Indeed, it is ambiguous as to which quality characteristics are more crucial for practitioners and how practitioners deal with those quality characteristics in their software development. So, in this study, a survey has been conducted on a group of practitioners who are involved in the software development for various industries. The goal of this survey is to understand which software quality characteristics are popular in industries and any modeling languages and tools used for dealing with the software quality.

2 RESEARCH METHODOLOGY

In our survey, we focus on the software quality characteristics determined by the ISO/IEC 25010:2011 SQuaRE software quality standard (ISO Central Secretary, 2011). As shown in Figure 1, ISO SQuaRE categorises the software quality into 8 different characteristics, which are functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability. Each software quality characteristic is also considered in terms of a cohesive set of quality sub-characteristics.

Our survey consists of 38 different questions, which are divided into four sections (i.e., one for each research question). Table 1 shows the list of questions. The questions 7-10 repeat for each software quality characteristic given in Figure 1. Also, some questions offer multiple-choice answers that allow the participants for choosing one or more answers. The multiple-choice questions with the free-text option essentially allow the participants to write their own answers that are not in the answer list. We evaluate the free-text answers and eliminate any of them that does not make sense for the question. Some questions require yes/no answers that are represented with five alternative options to maximise precision: always (100%), much of the time (>=75%), often (>=50%), sometimes (<50%), and never (0%).

The survey has been targeted for 16 different practitioners who have considerable experiences on software development and work for the Logo company[1], which is one of the largest software development companies in Turkey and has been offering various software solutions and services to different industries in Turkey for more than 30 years. To enhance the precision of the data gathered from the participants, each participant has been invited to a 30-60 minutes long session, in which one of us has been ready to interact with the participant. So, the participant has been requested to fill in the survey during the session and ask us any question that he/she may have about the survey questions. Also, the participants have been supplied with the ISO SQuaRE specification document so as to find out the precise definitions of the quality characteristics and their sub-characteristics if they need so.

Before executing the survey, we performed a plot study to get some initial feedback about the survey structure and questions. To this end, four different practitioners who have got 10+ years of experiences on the software development in IT industry shared their thoughts. So, this let us determine the missing or ambiguous answers, ambiguous questions, the duration of the survey sessions, and any supplementary materials for the participants taking the sessions.

3 RESEARCH QUESTIONS

To meet the goal stated in Section 1, a set of research questions has been determined. To understand how each research question is targeted, the survey ques-

[1]Logo web-site: https://www.logo.com.tr/en/

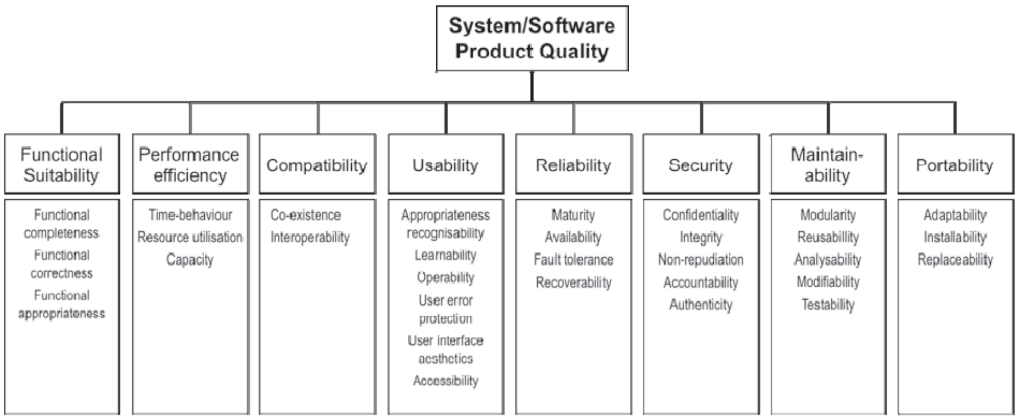

Figure 1: The software quality categorisation of the ISO SQuaRE software quality standard (ISO Central Secretary, 2011).

Table 1: The survey questions.

Res. Ques.	Survey Questions	Multiple Answers	Free Text	Yes/No Question	Integer Answer
RQ1	1- What is (are) your current job position(s)?	Yes	Yes	No	No
	2- What is(are) the type(s) of the software project(s)?	Yes	Yes	No	No
	3- How many years of experience do you have in software development?	No	No	No	Yes
	4- Which industries do your customers work in ?	Yes	Yes	No	No
RQ2	5- Which of the following software quality characteristics do you consider in software development projects ?	Yes	No	No	No
	6- If you do not consider any of those quality characteristics, please tell us the reason(s).	No	Yes	No	No
The questions 7-10 repeat for each software quality characteristic considered.					
RQ3	7- Can you rate for each software quality property how frequently you consider that property in your software projects?	No	No	Yes	No
RQ4	8- If you use model-driven approaches for modeling, analysing, and implementing the software systems for the quality properties of the current quality characteristics, please tell us about them.	No	Yes	No	No
	9- If you use any software tools for analysing the software systems for the quality properties of the current quality characteristic, please tell us more about them.	No	Yes	No	No
	10- If you manually analyse the quality properties of the current quality characteristic, please tell us more about how you perform the manual analysis	No	Yes	No	No

tions that are used for answering the research questions are shown in Table 1.

RQ1 - What Are the Profiles of the Participants? This research question aims to understand the participants' profiles including their job positions, the types of software projects involved, years of experiences, and work industries.

RQ2 - Which Software Quality Characteristics Do Practitioners Consider in Their Software Development? This research question aims to understand which of the software quality characteristics proposed by ISO SQuaRE are considered by practitioners in their software development projects.

RQ3 - Which Sub-characteristics of the Software Quality Characteristics Do Practitioners Consider in Their Software Development? The aim herein is to understand for each software quality characteristic of ISO SQuaRE which of its sub-characteristics are considered by practitioners for the quality analysis of their software systems.

RQ4 - Which Languages and Tools Do Practitioners Use for Analysing Their Software Quality? The aim here is to learn the software modeling languages that practitioners may be using for modeling their software systems and checking their software models for the quality properties of interest via the language toolset. Also, any commercial-off-the-shelf (COTS) tools that practitioners may be using for any quality characteristics are aimed to be learned.

4 THE SURVEY RESULTS

4.1 Profile Questions

Figure 2 shows the job positions of the the participants. So apparently, the design team lead and analyst

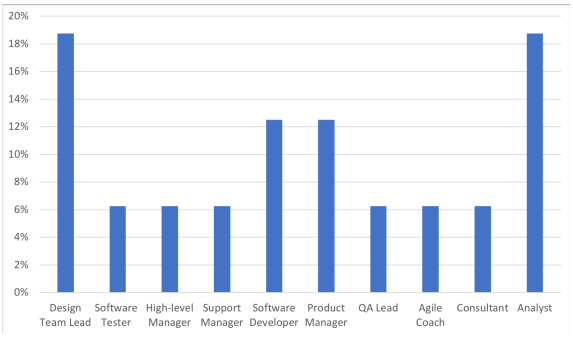

Figure 2: The participants' job positions.

positions have been the top selected ones, which are followed by the software developer and product manager positions. The rest of the job positions shown in Figure 2 are not that popular among the participants.

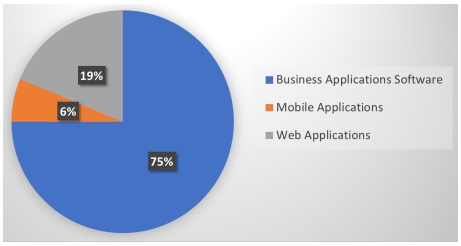

Figure 3: The types of the software projects that the participants are involved in.

Figure 3 shows the different types of software projects that the participants are involved in. So, most participants develop business applications software, which is followed by those developing web applications. A few participants stated that they develop mobile applications. The other types of software projects, such as safety-critical and mission-critical software systems and systems software are not in the scope of the participants.

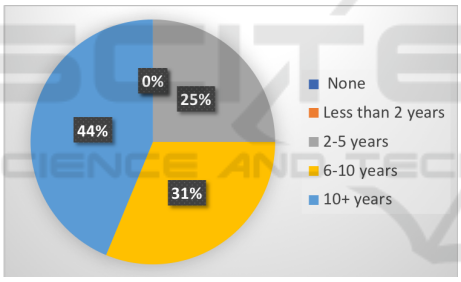

Figure 4: The participants' years of software development experiences.

Figure 4 shows the participants' years of experiences on the software development. So apparently, almost half of the participants have got 10+ years of experiences, 31% of them have 6-10 years of experiences, and the rest have 2-5 years of experiences.

Nearly all the participants develop software systems for various industries, which include automative and transportation, consumer electronics, defense/military & aviation, finance and accounting, government, healthcare and biomedical, IT and telecommunications, and software outsourcing. One of the participants stated that they work for the government only and another one work for the defense/military & aviation only.

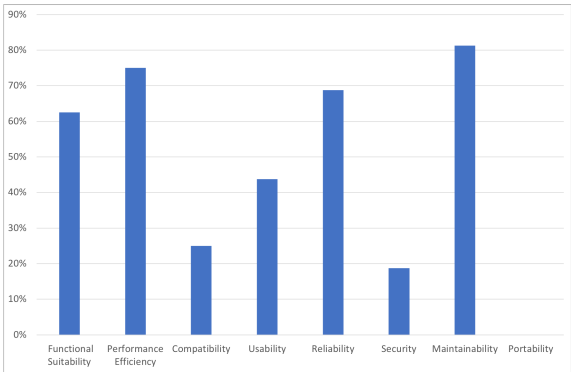

Figure 5: Participants' frequency of consideration for each software quality characteristic.

4.2 Software Quality Characteristics

Figure 5 shows the participants' consideration for the ISO SQuaRE quality characteristics in their software projects. So, maintainability is the top-selected characteristic, followed by the performance efficiency, reliability, and functional suitability characteristics. The compatibility and security characteristics are not so popular among the participants, and portability has not been considered by any of the participants at all.

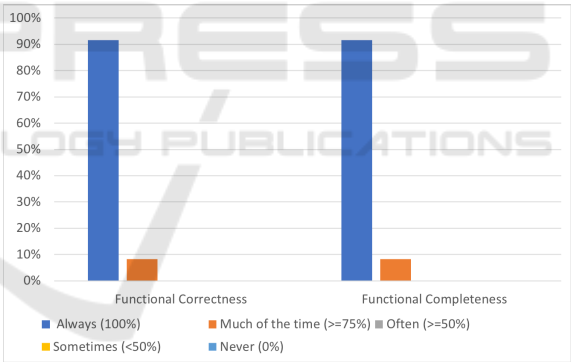

Figure 6: Participants' considerations for the functional suitability sub-characteristics.

4.2.1 Functional Suitability

The functional suitability quality characteristic is concerned with how well the provided functionalities of a software system meet the user expectations. The functional suitability is considered in terms of the functional correctness and functional completeness sub-characteristics. The functional correctness is to do with executing the system functionalities and checking that the obtained results are correct with regard to the system specifications. The functional completeness is to do with checking that the system functionalities cover all the required tasks.

Figure 6 shows the participants' frequency of consideration for the functional correctness and completeness. So, most of the participants always (100%) consider the functional correctness and completeness in their software projects.

As the survey results reveal, the participants use the Behaviour-Driven Development (BDD) methodology (Solis and Wang, 2011) for developing and testing software systems for the functional suitability. With BDD, different stakeholders (e.g., customers, users, and developers) interact to have the same understanding of the system functionalities and how each functionality is expected to behave. Then, the test scenarios are defined together with the users and customers in a natural language. The informal test scenarios can be translated into the executable test cases and the systems can be run for those test cases. To apply BDD in their software projects, the participants use the Cucumber framework[2].

The survey results also indicate that the participants have never used any software modeling languages for the modeling and analysis of the functional suitability properties.

4.2.2 Performance Efficiency

The performance efficiency quality characteristic is concerned with how much resources are consumed by a software system. The performance efficiency is considered in terms of the time behaviour, resource utilisation, and capacity sub-characteristics. The time behaviour is to do with ensuring that the time spent by the software system (e.g., sending a response or performing a particular operation) or the system throughput time meet the system requirements. The resource utilisation is to do with ensuring that the resources which the software system consumes meet the system requirements. The capacity is to do with ensuring that the upper limits that system functionalities can reach meet the system requirements.

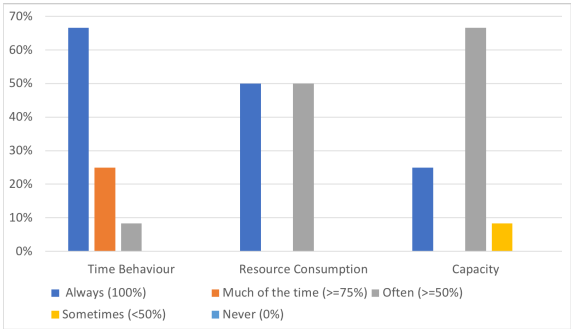

Figure 7: Participants' considerations for the performance efficiency sub-characteristics.

[2]https://cucumber.io/

Figure 7 shows the participants' considerations of the performance sub-characteristics in their software projects. While all the time behaviour, resource consumption, and capacity are frequently (i.e., always or much of the time) considered by most of the participants, the time behaviour attracted the greatest interest. Capacity is relatively less popular than the other two sub-characteristics – just 25% of the participants always (100%) consider the capacity analysis in their software project.

The participants do not use any software modeling languages for modeling and analysing the software systems against the performance properties. While some participants use their own software solutions for analysing performance, some use the COTS tools. These tools include dotMemory[3] for analysing the memory usage, dotTrace[4] for analysing the time behaviour, and Apache JMeter[5] for analysing capacity.

4.2.3 Compatibility

The compatibility quality characteristic is concerned with to what extent a software system can share resource(s) with another software system or work together with different systems without the need for any interventions. The compatibility is considered in terms of the co-existence and interoperability sub-characteristics. The co-existence property is to do with the ability of multiple software systems to share the same resources or execution platforms successfully. The interoperability property is to do with the ability of multiple software systems to interact with each other and exchange data successfully.

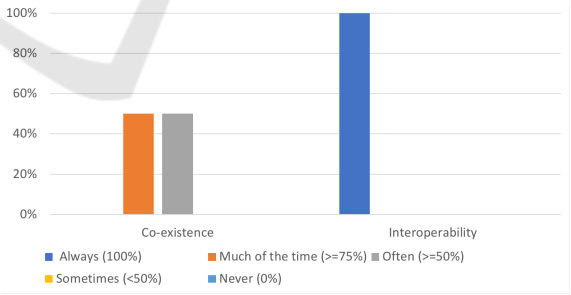

Figure 8: Participants' considerations for the compatibility sub-characteristics.

Figure 8 shows the participants' considerations of the co-existence and interoperability sub-characteristics in their software projects. So, while interoperability is always (100%) considered by the

[3]https://www.jetbrains.com/dotmemory/
[4]https://www.jetbrains.com/profiler/
[5]https://jmeter.apache.org/

participants, co-existence is not that popular essentially.

The participants do not use any software modeling languages for the modeling and analysis of their software systems and the compatibility sub-characteristics. However, the participants use their in-house software solutions for the analysis of the executable software systems for compatibility.

4.2.4 Usability

The usability quality characteristic is concerned with to what extent the potential users of a software system are capable of using the system in a way that meets the system requirements. Thu usability is considered in terms of the appropriateness recognisability, learnability, operability, user error protection, user interface aesthetics, and accessibility sub-characteristics. The appropriateness recognisability is to do with the users' acceptance of the software system for their needs. The learnability is to do with how easy it is to learn and use the software system. The operability is to do with how easy it is to operate the software system while the system is in use. The user error protection is to do with to what extent the software system prevents the users from making errors while using the system. The user interface aesthetics is to do with how attractive the user interfaces of the software system are to the users. The accessibility is to do with to what extent the software system can be used by the users with different types of disabilities.

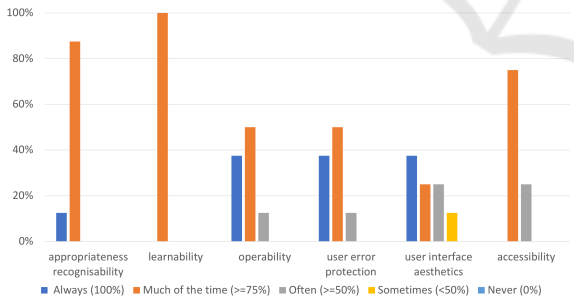

Figure 9: Participants' considerations for the usability sub-characteristics.

Figure 9 shows the participants' considerations of the different usability sub-characteristics in their software projects. So, all the participants frequently (i.e., much of the time or always) consider analysing the appropriateness recognisability and learnability. While operability and user error protection are also frequently considered by many participants, a few of the participants show less interest to them. Concerning the user interface aesthetics and accessibility, some participants do not frequently consider those two usability sub-characteristics in their software development. Indeed, the number of the participants who chose often or sometimes are the highest for the user interface aesthetics and accessibility.

None of the participants use any software modeling languages for the modeling and analysis of software systems and their usability characteristics. The participants' companies have user-experience (UX) teams who are responsible for testing the usability of software systems. Moreover, one of the participants stated that they use the A/B testing (Dixon et al., 2011) to analyse their systems' usability.

4.2.5 Reliability

The reliability quality characteristic is concerned with how well a software system performs its functionalities in its environment within the expected amount of time. The reliability is considered in terms of the maturity, availability, fault tolerance, recoverability sub-characteristics. The maturity is to do with how long the software system could be used in a way that satisfies the reliability requirements. The availability is to do with to what extent the software system is available for use when requested by the users. The fault tolerance is to do with to what extent the software system carries on operating in the case of any faults occuring. The recoverability is to do with to what extent the software system maintains its state without losing data in the case of any crashes due to some failures.

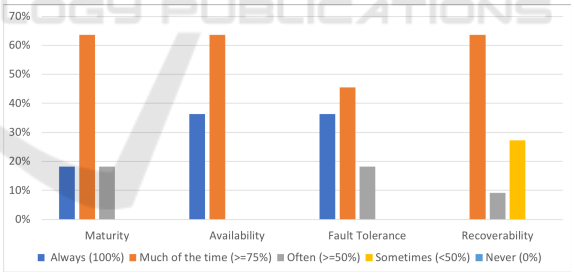

Figure 10: Participants' considerations for the reliability sub-characteristics.

Figure 10 shows how frequently the participants consider the reliability sub-characteristics in their software projects. So, all the participants frequently (i.e., either always or much of the time) analyse the availability of their software systems. While the maturity and fault tolerance are also frequently analysed by many participants, some of the participants often (>=50%) consider them. The recoverability property is shown quite less interest by the participants.

Concerning the languages and tools used, the participants do not use any languages or tools for the modeling and analysis of software systems and their reliability sub-characteristics.

4.2.6 Security

The security quality characteristic is concerned with to what extent a software system protects its data from malicious attacks so that the system data can only be accessed by the specified systems and people at the specified levels. The security quality characteristic is considered in terms of the confidentiality, integrity, non-repudiation, accountability, and authenticity sub-characteristics. The confidentiality is to do with ensuring that the system data is accessible by the authorised users only. The integrity is to do with to what extent the software system protects its data against the modifications performed by unauthorised users. The non-repudiation is to do with proving for the communication of any two parties that they cannot deny any event/action occuring between each other such as the message/data exchange. The accountability is to do with to what extent the actions taken by the software component or a person using the software system can be monitored to avoid any unwanted access to the system data or service. The authenticity is to do with whether it is possible to prove the identity of the users and any other entities or not.

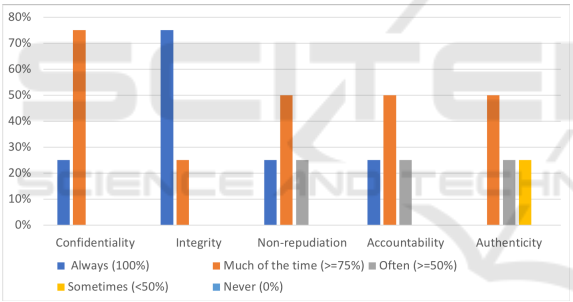

Figure 11: Participants' considerations for the security sub-characteristics.

Figure 11 shows the participants' considerations of the security sub-characteristics in analysing the software security. So, integrity is the most frequently considered sub-characteristic, which is followed by confidentiality. While non-repudiation, accountability, and authenticity properties are also frequently considered, some participants consider them infrequently (i.e., often or sometimes). Authenticity is essentially the least popular security sub-characteristic, as some participants stated to sometimes (<50%) analyse the authenticity of their software systems.

The participants do not use any software modeling languages for modeling and analysing the security sub-characteristics. However, the participants use the CheckMarx tool[6] for analysing their software system implementation against some security issues.

[6]https://www.checkmarx.com/

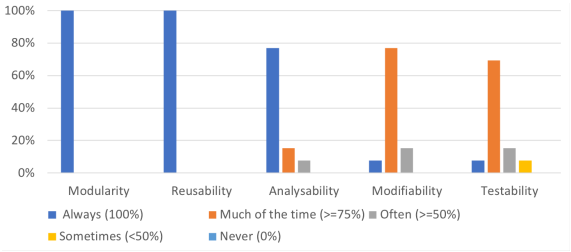

Figure 12: Participants' considerations for the maintainability sub-characteristics.

4.2.7 Maintainability

The maintainability quality characteristic is concerned with to what extent a software system can be modified, repaired, and improved effectively. The maintainability is considered in terms of the modularity, reusability, analysability, modifiability, and testability sub-characteristics. The modularity is to do with how well the software system is decomposed into modules that are independent from each other and each represent the cohesive set of functionalities of the system. The reusability is to do with to what extent the modules composing the software system can be re-used within the same system or across different systems. The analysability is to do with whether the software system can be analysed at any levels of abstractions for detecting any issues about the system. The modifiability is to do with to what extent the software system can be modified effectively without causing new errors in the system that require another modification(s). The testability is to do with at what level of coverage the test scenarios for the software system can be determined and the effective use of the test scenarios for testing the software system.

Figure 12 shows the participants' considerations of the maintainability sub-characteristics in their software projects. Modularity and reusability are the top-considered maintainability sub-characteristics by the participants, which are followed by the analysability. The modifiability and testability sub-characteristics are relatively less popular among the participants.

The participants do not use any software modeling languages for the maintainable software development. The participants use the SonarQube analysis tool[7] for analysing the software implementation for several maintainability issues.

4.2.8 Portability

The portability quality characteristic is concerned with to what extent a software system can be used on

[7]https://www.sonarqube.org/

another operating system or hardware. The portability is considered in terms of the adaptability, installability, and replaceability sub-characteristics. The adaptability is to do with to what extent the software system can be adapted to work in a different hardware or software environment. The installability is do with to what extent the software system can be installed/uninstalled successfully in/from any desired environments. The replaceability is to do with to what extent the software system or its components can be replaced with another system that can exhibit the same functionality.

Portability is just considered by one of the participants who stated to use the Jerkins tool for dealing with the portability issues.

5 SUMMARY OF THE RESULTS

In this section, the key outcomes obtained from the analysis of the survey results are summarised.

The Design Team Lead and Analyst Positions Are Highly Popular. While the participants represent many different job positions (e.g., software developer, consultant, manager positions, and tester), the design team lead and analyst are the top-selected job positions by the participants.

Business Applications Software Is the Top Popular Software Project Type. Most of the participants are involved in the business applications software development (75%). Concerning the rest, 19% are involved in the web applications development and 6% involved in the mobile applications.

Most Practitioners Have at Least 6 Years of Experiences on Software Development. While 44% of the participants have 10+ years of experiences, 31% have 6-10 years of experiences, and the rest have 2-5 years of experiences.

Maintainability and Performance Efficiency Are the Top Popular Quality Characteristics. Among the software quality characteristics proposed by ISO SQuaRE, most of the participants (>70%) consider the maintainability and performance efficiency characteristics for the analysis of their software systems. None of the participants (except one) are interested in analysing the software portability.

The Functional Suitability Sub-characteristics Are Equally Important. Almost all the participants who are interested in functional suitability always (100%) analyse the functional correctness and functional completeness of their software systems.

The Time Behaviour Is the Top Considered Performance Sub-characteristic. While most of the participants always (100%) analyse their software systems for the time behaviours, the resource consumption and capacity performance sub-characteristics did not attract that level of interest.

The Interoperability Is the Top Considered Compatibility Sub-characteristic. While the participants who are interested in compatibility always (100%) analyse their software systems for interoperability, those participants do not consider the co-existence sub-characteristic of compatibility so frequently.

The Learnability and Appropriateness Recognisability Are the Top Considered Usability Sub-characteristics. The participants frequently (i.e., always (100%) or much of the time(>=75%)) analyse the learnability and appropriateness recognisability of their software systems. While operability and user error protection are also quite frequently considered, the user interface aesthetics and accessibility sub-characteristics of usability are not so popular.

The Availability Is the Top Considered Reliability Sub-characteristic. The participants who are interested in reliability frequently analyse their software systems for availability, and that is followed by the analysis of the maturity and fault tolerance sub-characteristics. Recoverability is shown relatively less interest by the participants.

Confidentiality and Integrity Are the Top Considered Security Sub-characteristics. The participants who are interested in security frequently analyse their software systems for confidentiality and integrity. While non-repudiation and accountability are also quite popular, authenticity is shown relatively less interest by the participants.

Modularity and Reusability Are the Top Considered Maintainability Sub-characteristics. The participants who are interested in maintainability always consider the modularity and reusability of their software systems. The analysability of software systems is also quite popular. However, modifiability and testability are shown less interest by the participants.

None of the Participants Use Any Modeling Languages for Analysing The Software Quality Early On. While many software modeling languages have been existing today for the modeling and analysis of software systems for different quality characteristics, the participants are not familiar with any of them.

The Participants Use Cots Tools for the Software Analysis. The participants have been observed to use several COTS tools available for analysing the software system implementations for the quality characteristics of interest. While some of these tools are free and open-source, some are commercial tools.

6 RELATED WORK

While the literature includes many survey studies on the software quality, it is really hard to find any surveys that focus on various types of quality characteristics in general and give any results regarding the perspectives of the practitioners who are involved in the software development.

In (Pérez et al., 2013), the authors targeted the Belgian companies and surveyed 44 different participants working in different companies to understand the processes, tools, and techniques that they employ for improving the software quality in general. However, Pérez et al. do not focus on any software quality characteristics and practitioners' perspectives on them, as is the case in our survey. In (Garousi and Zhi, 2013), the authors surveyed 245 practitioners who work in Canada and aimed at understanding the techniques, tools, and metrics used for testing software systems. In (Bygstad et al., 2008), the authors surveyed 78 different IT companies in Norway to understand their perspectives towards the software usability. In (Gulliksen et al., 2004), the authors surveyed 194 different practitioners who work in Sweden to understand the techniques and tools that they use for promoting the usable software systems and their perspectives towards maximising the software usability. In (Geer, 2010), the author surveyed among 46 different practitioners who hold either the consultant, manager, or developer positions, and intended to understand their perspectives towards the secure software development life-cycles. In (Moores and Edwards, 1992), the author surveyed 54 different large IT companies that are located in United Kingdom and aimed to understand to what extent the practitioners in those companies use the cost estimation tools for the purpose of estimating the size and cost of their software development projects. In (Sousa and Moreira, 1998), the authors conducted a survey among the 37 different organisations located in Portugal. Sousa et al. aimed at understanding how the software maintenance process is carried out and the difficulties encountered. In (Xie et al., 2011), the authors interviewed 15 different software developers to basically understand the software security issues that they encountered in their software projects, the methods, techniques, and tools that they use for handling the software security issues, and their opinions about how the software security relates to the software development life-cycle.

7 CONCLUSION

In this paper, a survey has been conducted on understanding the practitioners' perspectives on different software quality characteristics and the modeling languages/tools used for analysing the software quality. To this end, the survey focuses on the ISO SQuaRE software quality standard, which categorises the software quality as the functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability and portability characteristics. Each software quality characteristic is defined by ISO in terms of a relevant set of sub-characteristics. The survey consists of 38 questions that have been answered by 16 participants who work for a Turkish software company called Logo. Each participant has been allocated a 30-60 minutes session during which the participant has been requested to fill in the survey and allowed to ask any questions that he/she has about the survey questions. The analysis of the survey results lead to many interesting lessons. Practitioners give the highest importance to *(i)* the maintainable development of software systems that can be easy to change and improve after deployment and *(ii)* the efficient use of the resources. The portable software development for facilitating the use of the software system in different environments is not shown the least interest by the practitioners. Practitioners also showed varying levels of interests to the different sub-characteristics of the software quality characteristics proposed by ISO SQuaRE, and that led to determine practitioners' priorities regarding the analysis of software systems for each quality characteristic. Moreover, practitioners do not use any software modeling languages for the high-level modeling and analysis of software systems. Note that many modeling languages have been actually existing (e.g., AADL, UPPAAL, and JML), which enable the exhaustive model checking of the software models and prove the models for different quality properties of interest before proceeding with the software implementation. Practitioners prefer to use the COTS tools available on the market that allow for checking the software implementation for some quality characteristics. However, any quality issues detected may not easily be fixed at this stage, as the software systems analysed have already been design and implemented.

Concerning the threats to the validity of the survey results, the participants have been selected from a single company in Turkey. While this may cause biases regarding the selection of the participants, the Logo company essentially develops software systems for various customers of different industries in Turkey and that helps addressing the software quality require-

ments of customers with different needs. Also, some software quality characteristics addressed in the survey may not be understood precisely by all the participants, and therefore, the survey has been decided to be conducted via sessions in which the participants are requested to fill in the survey and allowed to ask for any clarifications. Lastly, the survey has been conducted with 16 different participants only. While this essentially helped us to get an initial idea about the research questions targeted, we plan to conduct a larger survey with similar research questions in the future.

REFERENCES

Bygstad, B., Ghinea, G., and Brevik, E. (2008). Software development methods and usability: Perspectives from a survey in the software industry in norway. *Interacting with Computers*, 20(3):375–385.

Cavano, J. P. and McCall, J. A. (1978). A framework for the measurement of software quality. *SIGSOFT Softw. Eng. Notes*, 3(5):133–139.

Chalin, P., Kiniry, J. R., Leavens, G. T., and Poll, E. (2005). Beyond assertions: Advanced specification and verification with JML and ESC/Java2. In de Boer, F. S., Bonsangue, M. M., Graf, S., and de Roever, W. P., editors, *FMCO*, volume 4111 of *Lecture Notes in Computer Science*, pages 342–363. Springer.

Çiflikli, B., Özcan, O., and Uysal, A. B. (2012). Model driven software development for military command and control systems. In *2012 IEEE 36th Annual Computer Software and Applications Conference*, pages 494–497.

Dixon, E., Enos, E., and Brodmerkle, S. (2011). A/b testing of a webpage. US Patent 7,975,000.

Feiler, P. H., Lewis, B. A., and Vestal, S. (2006). The SAE architecture analysis & design language. In *IEEE Intl Symp. on Intell. Control*, pages 1206–1211. www.aadl.info.

Garousi, V. and Zhi, J. (2013). A survey of software testing practices in canada. *Journal of Systems and Software*, 86(5):1354–1376.

Garvin, D. (1984). What does product quality really mean? *Sloan Management Review*, 26:25–45.

Geer, D. (2010). Are companies actually using secure development life cycles? *Computer*, 43(6):12–16.

Gulliksen, J., Boivie, I., Persson, J., Hektor, A., and Herulf, L. (2004). Making a difference: a survey of the usability profession in sweden. In Raisamo, R., editor, *Proceedings of the Third Nordic Conference on Human-Computer Interaction 2004, Tampere, Finland, October 23-27, 2004*, pages 207–215. ACM.

Haghighatkhah, A., Oivo, M., Banijamali, A., and Kuvaja, P. (2017). Improving the state of automotive software engineering. *IEEE Software*, 34(5):82–86.

ISO Central Secretary (2011). Systems and software engineering – systems and software quality requirements and evaluation (square) – system and software quality models. Standard ISO-IEC 25010:2011, International Organization for Standardization, Geneva, CH.

Kan, S. H. (2014). *Metrics and Models in Software Quality Engineering - Paperback*. Addison-Wesley Professional, 2nd edition.

Kitchenham, B. A. and Pfleeger, S. L. (1996). Software quality: The elusive target. *IEEE Software*, 13(1):12–21.

Larsen, K. G., Pettersson, P., and Yi, W. (1997). UPPAAL in a nutshell. *STTT*, 1(1–2):134–152.

Marques, J. and Cunha, A. (2013). A reference method for airborne software requirements. In *2013 IEEE/AIAA 32nd Digital Avionics Systems Conference (DASC)*, pages 1–29.

Moores, T. T. and Edwards, J. S. (1992). Could large uk corporations and computing companies use software cost estimating tools? – a survey. *European Journal of Information Systems*, 1(5):311–320.

Nitze, A. and Schmietendorf, A. (2015). A survey on mobile users' software quality perceptions and expectations. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 1–2.

Ozkaya, M. (2018). The analysis of architectural languages for the needs of practitioners. *Softw., Pract. Exper.*, 48(5):985–1018.

Pérez, J., Mens, T., and Kamseu, F. (2013). A pilot study on software quality practices in belgian industry. In *2013 17th European Conference on Software Maintenance and Reengineering*, pages 395–398.

Richardson, I., Reid, L., and O'Leary, P. (2016). Healthcare systems quality: Development and use. In *2016 IEEE/ACM International Workshop on Software Engineering in Healthcare Systems (SEHS)*, pages 50–53.

Sajić, M., Bundalo, D., Bundalo, Z., and Pašalić, D. (2017). Digital technologies in transformation of classical retail bank into digital bank. In *2017 25th Telecommunication Forum (TELFOR)*, pages 1–4.

Solis, C. and Wang, X. (2011). A study of the characteristics of behaviour driven development. In *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 383–387.

Sousa, M. J. C. and Moreira, H. M. (1998). A survey on the software maintenance process. In *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*, pages 265–274.

Tian, J. (2005). *Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement*. Wiley-Interscience, New York, NY, USA.

Xie, J., Lipford, H. R., and Chu, B. (2011). Why do programmers make security errors? In *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 161–164.