Software Quality Journal (2022) 30:983–1002
https://doi.org/10.1007/s11219-022-09589-y

Quantum service-oriented computing: current landscape and challenges

Enrique Moguel[1] · Javier Rojo[1] · David Valencia[1] · Javier Berrocal[1] · Jose Garcia-Alonso[1] · Juan M. Murillo[1]

Accepted: 20 March 2022 / Published online: 25 April 2022

Abstract
The development that quantum computing technologies are achieving is beginning to attract the interest of companies that could potentially be users of quantum software. Thus, it is perfectly feasible that during the next few years hybrid systems will start to appear integrating both the classical software systems of companies and new quantum ones providing solutions to problems that still remain unmanageable today. A natural way to support such integration is Service-Oriented Computing. While conceptually the invocation of a quantum software service is similar to that of a classical one, technically there are many differences and technological limitations, which refer to platform independence, decoupling, scalability, etc. To highlight these differences and the difficulties to develop quality quantum services, this paper takes a well-known problem to which a quantum solution can be provided, integer factorization, making use of the Amazon Braket quantum service platform. The exercise of trying to provide the factorization as a quantum service is carried out following the best practices, design patterns and standards existing in the implementation of classical services. This case study is used to highlight the rough edges and limitations that arise in the integration of classical-quantum hybrid systems using service-oriented computing. The conclusion of the study allows us to point out directions in which to focus research efforts in order to achieve effective quantum service-oriented computing.

Keywords Quantum services · Classical services · Hybrid classical-quantum software · Quality

1 Introduction

Quantum computing is starting to establish itself as a commercial reality (MacQuarrie et al., 2020). Several major computing corporations have already built working quantum computers, there are tens of quantum programming languages and simulators, and real quantum computers can already be used by the general public through the cloud. All this is motivating software development companies to take the first steps by launching their own

✉ Enrique Moguel
enrique@unex.es

Extended author information available on the last page of the article

proposals for the integral development of quantum software (Pérez-Castillo & Piattini, 2020; Wille et al., 2019; Bergholm et al., 2018; Piattini et al., 2021; Pérez-Castillo et al., 2021). All of these signals are an urgent call to software engineers to prepare and enroll to sail the quantum seas.

It is generally assumed that on the way to a new world in which software systems are mostly quantum, there will be a transition time in which classical and quantum systems must not only coexist but collaborate by interacting with each other (Sodhi, 2018). This is what has been called classical-quantum hybrid systems (McCaskey et al., 2018, 2020). The advances provided by software engineering in the last two decades allow us to affirm that a natural way to approach such collaborative coexistence is by following the principles of service engineering and service computing.

Among the reasons for this, two can be highlighted. On the one hand, as hardware technology matures and achieves more affordable costs, it is reasonable to think that companies will be inclined to use quantum infrastructure and quantum software as a service, as they are used to do nowadays with classical computing resources. This has already happened with classical computing services, companies such as Amazon, Microsoft, IBM, and Google that have started approaching the world of quantum computing Digital Journal (2022). And it is not unreasonable to think that these same companies will offer both classical and quantum computing services indiscriminately.

On the other hand, it is reasonable to think that at least initially, quantum systems will be used to solve only those parts of problems that cannot be solved by classical architectures, while those parts of problems that are already efficiently solved by classical architectures will continue to be treated as before. For example, in the field of health, it will be possible to accelerate the discovery of new medicines, perform simulations of molecules for pharmaceuticals, and enable the development of new medicines easier and faster (Zinner et al., 2022); in the financial domain, it could help analyze all possible scenarios and compare risks and optimize a financial portfolio Pistoia et al. (2021); it could also help us decrypt cryptographic security systems, optimize travel routes and logistics, model climate change, etc. (Cheng et al., 2021).

A natural way to achieve these quantum solutions is by consuming quantum services.

Conceptually, the invocation of a quantum program is similar to that of a classical service. A piece of software needs a result to be produced by a quantum system and to do so it consumes a service. For the sake of service engineering principles, such an invocation should even be agnostic of whether the service that will return the result is quantum or not. Technically, however, the invocation of a quantum service is very different from that of classical service and still poses a challenge today. This is due to the inherent nature of quantum computing, meaning that a quantum service differs from classical services in which it includes entanglement and superposition of solutions, and will collapse to a single solution when interacting with the external world, leading to having a probability amplitude associated to the results obtained upon observations of the quantum system.

Servitizing a quantum piece of software, namely converting it into a service endpoint that can be invoked through a standard service request, is possible with the existing technology. However, in the current status of quantum software, it means eliminating most of the advantages that made service-oriented computing a commercial success, especially, those related to software quality like composability, modularity, maintainability, reusability, etc. (Ravichandran & Rai, 2000).

The reasons for this are multiple. First and foremost, the specificity of each architecture makes quantum algorithms and their parameters dependent on the specific quantum hardware in which they will be executed. But also, the return of the result of a quantum process is subject to errors or does not support the intermediate verification of results (due to the system collapse). Thus, different quantum architectures require very different skillsets. For example, circuit-based quantum programming requires developers to know the details of quantum gates (Wille et al., 2019), while quantum annealing programming requires adapting the problem to that specific metaheuristic (Boixo et al., 2013). Consequently, invoking a quantum program in an agnostic way is impossible today and violates all the principles of service engineering. All of the above highlights the need for the development of Quantum Service Engineering (Piattini et al., 2020).

In this paper, we explore the current state of quantum software engineering from a service-oriented point of view. The integer factorization problem (Nielsen & Chuang, 2002; Jiang et al., 2018) is used to illustrate the different problems that arise when a quantum piece of code is tried to be used as a service. Amazon Braket[1], the quantum computing service offered by Amazon as part of their AWS suite, is used as the services platform. Amazon is globally recognized as the leader company in services technology, and through Braket, they offer access to quantum computers from three different hardware providers. Using this platform as the basis for quantum services development, we identify the problems and limitations of current technology using the lessons learned from service-oriented computing. The paper provides an exploration of the problems to be addressed pointing out different research directions for the development of a future quantum service engineering.

In order to do that the rest of the paper is organized as follows. Section 2 details this work background in both fields of service-oriented computing and quantum software development. Section 3 describes one of the possible ways to implement a classic service following best practices, design patterns, and existing standards. Section 4 addresses the servitization of quantum software using Amazon Braket. Section 5 lists the main limitations found in today technology that limits the benefits of quantum services. And finally, Sect. 6 presents the paper conclusion and future works.

2 Background

Service-oriented computing is a paradigm that utilizes services as the fundamental elements for developing software (Papazoglou, 2003). One of its pillars is service-oriented architecture (SOA) that proposes the implementation of complex software solutions through the use of a set of services that are composed and choreographed (Endrei et al., 2004). The basic composition mechanism is the service call that allows a service to be invoked from another piece of code (potentially another service) agnostically with respect to the place, technology, or architecture of the invoked service. The services can thus be maintained, evolved, replaced, and reused independently without affecting the software that invokes them. It is precisely these properties that make them especially attractive to create quality software. Over the last two decades, service-oriented computing and SOA in general, and web services in particular, have been at the center of intense research (Bouguettaya et al., 2017) leading to monolithic software being gradually replaced by service-based software run in the cloud (Mazlami et al., 2017; Haugeland et al., 2021).

The success of service-oriented computing has been possible, to a great extent, thanks to the development of cloud computing as a paradigm that aims to provide reliable and

[1] https://aws.amazon.com/braket/

customized dynamic computing environments (Wang et al., 2010). Some of the main reasons behind the success of the cloud include: the ability for companies to better control their costs since they do not have to buy, upgrade and maintain expensive hardware and only pay for their use; and the flexibility and scalability provided by cloud vendors that allow companies to instantly increase or decrease their hardware capabilities according to their needs. These have made the cloud one of the most successful business models of the last decades. Recent estimations calculate that in the USA only, cloud computing contributed approximately 214 billion dollars in value-added to the GDP and 2.15 million jobs in 2017 Hooton (2019).

Given these numbers are not a surprise that current quantum computers, which are still very expensive hardware to build and operate, are being offered following this model. In its current form, most quantum computers can be accessed through the cloud in a model called by some researchers quantum computing as a service (QCaaS) (Rahaman & Islam, 2015). This model can be compared to the classical Infrastructure as a service (IaaS) model offered in cloud computing. QCaaS allows developers to access some of the world's existing quantum computers; nevertheless, this access is very dependent on the specific hardware and developers must have great proficiency in Quantum Computing to benefit from its advantages.

To increase the abstraction level of QCaaS, there are multiple ongoing research efforts. From a commercial perspective, platforms like the above-mentioned Amazon Braket provide a development environment for quantum software engineers or, like QPath[2], an ecosystem that covers a wide range of possible applications by integrating the software classical and quantum worlds in a quantum development and application life cycle platform for high-quality quantum software.

From a more academic perspective, a significant number of works are starting to appear in the field of quantum software engineering (Zhao, 2020; Piattini et al., 2020). These works focus on translating the lessons learned in classical software engineering to improve the quality of quantum software. However, as far as the authors know, very few works focus on the perspective of service engineering for quantum and hybrid software.

However, some works are starting to appear in this domain, like Barzen et al. (2021) where quantum application as a service (QaaS) is proposed to narrow the gap between classical service engineering and quantum software. Works like this reveal the need to focus on a service-oriented approach for the development of quantum services.

3 A good classic service implementation

Before starting to discuss the proposed case study and the limitations found, we will describe one of the possible ways to manage the entire life-cycle of a classic service, starting with the implementation, followed by its deployment and its subsequent monitoring and maintenance. For this purpose, we will follow existing best practices, design patterns, and standards.

Figure 1 shows how a good classical version of the implementation, deployment, and maintenance of a service could look like. We would like to point out that there is no single best way to define a classic service and that it can be developed following different

[2] https://www.quantumpath.es/

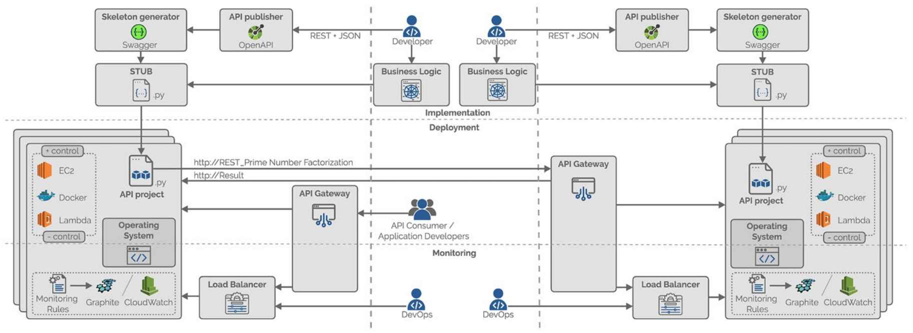


Fig. 1 A good classic version of a implementation of a service

approaches. However, we have tried to follow some of the most accepted best practices, design patterns, and standards that have become very relevant in recent years for the definition and development of classical services. Among them, we followed the models defined by Newman (2021) (one of the early pioneers of microservice architecture), the microservices design patterns of Richardson (2019) (a renowned and well-respected microservices expert), and the guide of Wolff (2019) (a renowned software architect who has written several books on microservices).

In this regard, Fig. 1 is split vertically in two. On the left side can be seen the service client (which can be a service itself). In the case study, which will be detailed in Sect. 4, this would be the cryptographic decryption service. And on the right side, the invoked service can be seen, in the commented case study, it would be the algorithm for calculating the factorization of integers.

Horizontally, the same Fig. 1 is split into three layers that represent different phases in the service life cycle. On the top, the implementation phase of the service can be seen, where the service is developed. In the middle, the deployment phase of the service can be seen, where it is published so that it can be invoked. And at the bottom, the monitoring phase can be seen where the service is being used. For each of these phases, some of the most relevant standards and best practices in classic services will be discussed.

For the implementation of a classical service, a developer needs to combine two main aspects. The business logic of the service, which is specific to each service and implemented ad hoc, and the service API. For the definition of the service API, the OpenAPI[3] specification can be used. This specification defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. An OpenAPI definition can then be used by code generation tools like Swagger to generate servers and clients in various programming languages. Using OpenAPI, for our case study, the integer factorization service can be defined with its input and output parameters. From this definition, a code stub can be generated, for example in Python, in which the business logic of the service can be added. By following this approach, the service could be accessed using a REST request and JSON to provide the input parameters.

3 https://www.openapis.org/

Once the service is implemented, it must be deployed so clients can access it. For this deployment, in most cases, a cloud computing solution is used in which we deploy our API project in hardware provided by a third party. Depending on the level of control, we require over the hardware, there are several alternatives ranging from IaaS solutions, like Amazon Elastic Cloud Computing (EC2)[4], where we have complete control over a virtual machine, to the use of containers like Docker[5] or to serverless approaches like Amazon Lambda[6], where no control over the underlying infrastructure is provided to the service creators. Independently of the deployment approach followed the proposed service will have some coupling with the operating system in which it is deployed.

At the same level, one of the most used design patterns in the deployment of classical services is the API Gateway. To put it simply, the API Gateway takes all API requests from a client, determines which services are needed, and combines them into a unified, seamless experience for the user. For the proposed case study, the service would invoke the prime number factorization service through the API Gateway of the back-end. The API Gateway would transmit the petition to the appropriate service and provide the response once it has been computed.

Finally, once the service is deployed and is being invoked by users it must be monitored by the DevOps team to ensure that the service can deal with the demand. To address this, usually, a Load Balancer can be used in combination with the API Gateway. Additionally, monitoring rules can be defined in combination with tools like Graphite[7] or CloudWatch[8] to obtain fine-grained information of the service status. This would allow us to control the status of the prime number factorization service, the number of requests, how much are we going to pay for the infrastructure at the end of the month, etc.

As can be seen, even for a “simple” service like the prime number factorization service the development, deployment, and monitoring process are quite complex. It could be thought that most of the components in Fig. 1 are not really necessary for a simple service. However, all of them are there for a reason. Service-oriented computing has adopted these patterns and good practices because all of them contribute to the benefits provided by service orientation. The most relevant of these benefits are:

- **Platform independence.** By using a REST API, which in turn is based on HTTP, IP, and the rest of internet protocols, services can be invoked from any platform independently of the language of the service, the language of the client, and the platforms in which both are run. Similarly, by using JSON, data can be transferred between services regardless of the implementation languages. Finally, by using the OpenAPI specification, an independent, well-formed API will be defined that can be easily consumed by any client.
- **Location independence.** Classical services are also independent of the location in which they are deployed. This independence can be considered at two levels:

[4] https://aws.amazon.com/ec2

[5] https://www.docker.com/

[6] https://aws.amazon.com/lambda

[7] https://graphiteapp.org/

[8] https://aws.amazon.com/cloudwatch

  - *Physical independence.* This is provided by REST, HTTP, and DNS which allow service developers to use a URL to access a service independent of where it is deployed.
  - *Logical independence.* Meaning the independence of a given service inside a more complex project is provided by an API Gateway that hides this complexity from the service clients.

- **Decoupling.** Three different types of decoupling must be taken into account:

  - *Decoupling between services.* The decoupling between different services of a single API is provided by the API Gateway which can communicate services without the services knowing each other.
  - *Decoupling between services and hardware.* The decoupling between services and hardware and operating systems is provided by the platform in which the services are deployed (EC2, Docker, Lambda, etc.).
  - *Decoupling between services and programming language.* The decoupling between the services and the programming language in which they are written is provided by the OpenAPI Specification and code generation tools that are able to generate the service stub in different programming languages.

- **Scalability.** Services are also resilient and developers are able to scale to address changes in the demand. The most common way to provide this scalability, without incurring high economic costs from the cloud provider, is the use of elastic platforms and a load balancer that is able to start and stop new instances of the service as needed.
- **Composability.** Services also have to be composable, so a set of simple services can be composed to provide solutions to complex problems. This composability can be provided by the API Gateway that can decompose a single invocation into calls to different services without the services knowledge.
- **Reliability.** Services have to be reliable in the broadest sense of the word, including security, maintainability, accountability, and many other X-abilities. To achieve this, a complete set of monitoring and analytic tools are needed that provide all the information needed about the services.

To achieve all these benefits a complex infrastructure, like the one shown in Fig. 1, is needed. However, as far as the authors know, there is no support to obtain the benefits of service-oriented computing if one of the services of an API is implemented as a quantum service.

4 Quantum servitization

There are some problems that cannot be solved efficiently using classical computation, and a clear example is that of factoring prime numbers. This is where quantum computing can offer the greatest benefits. But without forgetting all the knowledge acquired in classical service-oriented computing. Where being able to distribute and servitize a development has offered benefits already known to all.

To address the current state of quantum services, in this paper, we have decided to use Amazon Braket. Amazon defines Braket as a fully managed quantum computing service. Specifically, Braket provides a development environment to build quantum algorithms, test them on quantum circuit simulators, and run them on different quantum hardware technologies.

Given that Amazon is currently the global leader regarding cloud computing and services technologies through AWS, Braket seems a good alternative to develop quantum services. Nevertheless, since the state of quantum software development is roughly the same in the different existing platforms, we expect similar results to the ones presented in this paper if the quantum services were developed on a different platform.

The basic building block of service-oriented computing is a service, defined as a self-describing, platform-agnostic computational element that supports rapid, low-cost composition of distributed applications (Papazoglou, 2003). However, Braket is not directly prepared to offer the developed quantum algorithms as services that can be invoked through an endpoint to compose a more complex application.

This shortcoming can be addressed by wrapping the quantum algorithm in a classical service. This implies including a classical computer to run the classical service that, in turn, invokes the quantum computer. As far as the authors know, there is currently no way of directly invoking a quantum algorithm as a service. Figure 2 shows an example of this approach. One of the simplest and well-known quantum circuits, the one used to create Bell states between two qubits is wrapped by a Flask[9] service. This Flask service can be deployed in a classical computer and provides a simple way to include quantum algorithms in a complex service-oriented solution.

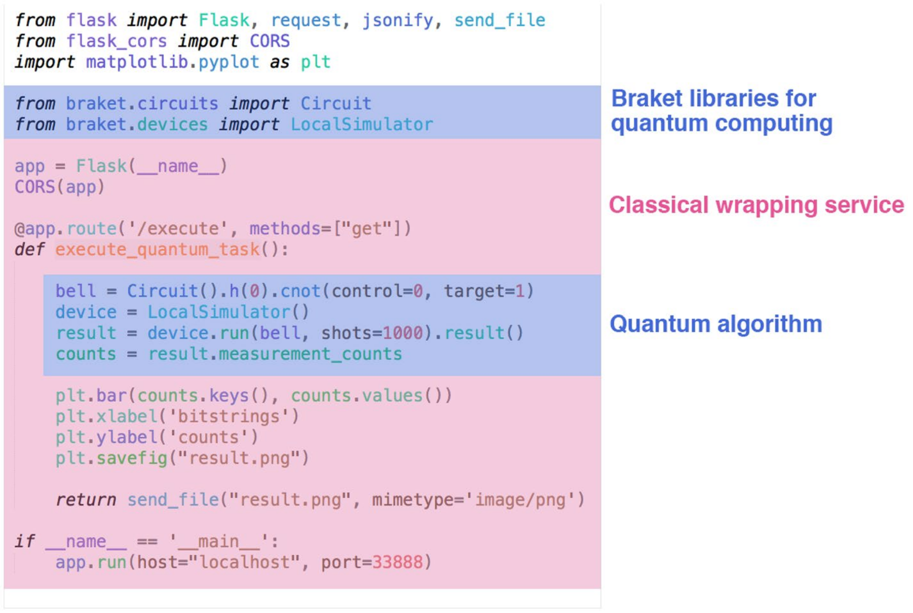
Next, we present a more complex quantum algorithm used as a case study to identify the problems and limitations of current technology from the perspective of Service-Oriented Computing.

4.1 Integer factorization case study

In order to make the analysis as broad and interdisciplinary as possible, we have decided to select a problem well-known by the scientific community working in quantum computing. At the same time, the selected problem is simple enough to be comprehended by any newcomer. Between the several applications that satisfy both conditions, we have decided to tackle Integer Factorization, more precisely with a particular application of the later denoted Prime Factorization. As we all know, although this fundamental problem in number theory is computationally hard, it is not believed to belong to the NP-hard class of problems (Jiang et al., 2018). Nonetheless, it is a problem that has been used as a basic hardness assumption for cryptographic algorithms, such as the famous RSA algorithm. Thus, integer factorization and identification of new methods to address this task acquire an important role in information security.

There are multiple proposals and algorithms for the solution of this problem, being the most famous Shor's algorithm Nielsen and Chuang (2002). This algorithm is normally described in terms of quantum gates and circuits, suitable for development and execution on machines such as IBM's Q computing chip Haring et al. (2011), but when considering other approaches to quantum computing, such as Adiabatic Quantum Computing based on concepts such as quantum annealing, it is not possible to implement Shor's algorithm directly. Nonetheless, other algorithms have been proposed for prime

[9] https://flask.palletsprojects.com/



```
from flask import Flask, request, jsonify, send_file
from flask_cors import CORS
import matplotlib.pyplot as plt

from braket.circuits import Circuit
from braket.devices import LocalSimulator

app = Flask(__name__)
CORS(app)

@app.route('/execute', methods=["get"])
def execute_quantum_task():

    bell = Circuit().h(0).cnot(control=0, target=1)
    device = LocalSimulator()
    result = device.run(bell, shots=1000).result()
    counts = result.measurement_counts

    plt.bar(counts.keys(), counts.values())
    plt.xlabel('bitstrings')
    plt.ylabel('counts')
    plt.savefig("result.png")

    return send_file("result.png", mimetype='image/png')

if __name__ == '__main__':
    app.run(host="localhost", port=33888)
```

Braket libraries for quantum computing
Classical wrapping service
Quantum algorithm

Fig. 2 Quantum algorithm wrapped by a classical service

factors, such as the case of the algorithm proposed by Wang et al. in (2020). Thus, in the studies conducted on this paper, these will be the algorithms proposed for integer factorization: Shor's algorithms for quantum machines programmed with quantum circuits and gates, such as Rigetti's Motta et al. (2020) and IonQ's Kielpinski et al. (2002); and integer factorization based on quantum annealing for adiabatic quantum machines such as D-Wave's Hu et al. (2019).

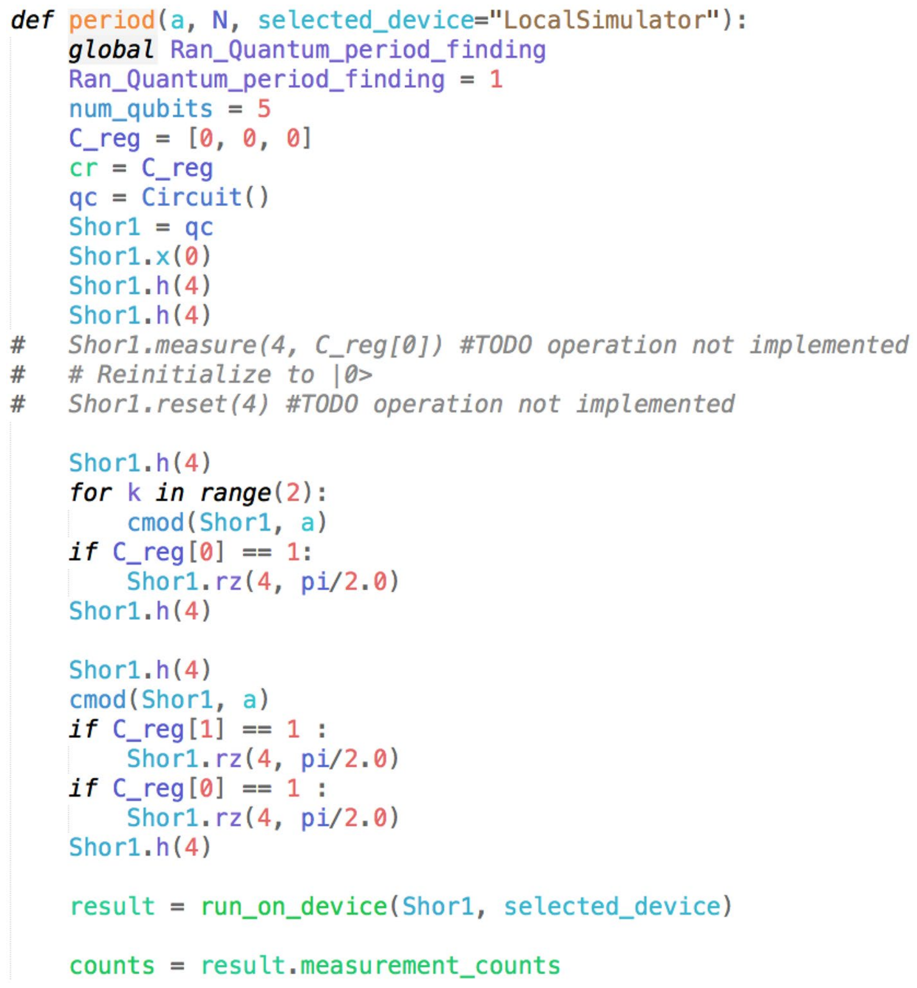
These algorithms also serve as an illustration of a problem derived from the relative novelty of quantum computing and its different existing implementations. Namely, the nonexistence of algorithms with do-it-yourself characteristics. This is mainly due to the complex nature of the problems addressed by quantum computing and to the proximity of the algorithms with the underlying hardware used. This context is producing problems similar to those of the 60s software crisis Moguel et al. (2020), where each algorithm was designed for each particular computing hardware, many times having to recreate the algorithms for each new machine or even for each new increment of the problem. A reminiscent of this is found, for example, when having to generate a new circuit in Shor's algorithm for primes to be factorized. Although this could be done through the use of algorithms to generate these circuits automatically, for the great majority of possible users of quantum computing, the ability to be able to create these types of "meta-algorithms" is beyond their capabilities, complicating the expansion of quantum computing usage out of the specialized field. Thus, it is necessary to offer solutions to non-specialized users for the utilization of quantum computing, such as the case of deployment of quantum services which allow hiding the complexity to users, only providing with entry end-points and returning the results of the execution.

4.2 Integer factorization in Amazon Braket

To illustrate the actual situation of quantum services that can be developed on Amazon Braket, we have translated the above-mentioned integer factorization algorithms to this platform.

At the moment of writing this manuscript, Braket supports three different quantum computer simulators and real quantum computers from three different hardware vendors. Specifically, the supported quantum computers include two vendors whose development is based on quantum circuits, Rigetti and IonQ, and one vendor based on quantum annealing, D-Wave. The integer factorization algorithms have been tested in all supported quantum machines and simulators.

Since the supported simulators are also based on quantum circuits, Shor's algorithm has been used in both, simulators and quantum circuits hardware. Figure 3 shows a fragment



```
def period(a, N, selected_device="LocalSimulator"):
    global Ran_Quantum_period_finding
    Ran_Quantum_period_finding = 1
    num_qubits = 5
    C_reg = [0, 0, 0]
    cr = C_reg
    qc = Circuit()
    Shor1 = qc
    Shor1.x(0)
    Shor1.h(4)
    Shor1.h(4)
#   Shor1.measure(4, C_reg[0]) #TODO operation not implemented
#   # Reinitialize to |0>
#   Shor1.reset(4) #TODO operation not implemented

    Shor1.h(4)
    for k in range(2):
        cmod(Shor1, a)
    if C_reg[0] == 1:
        Shor1.rz(4, pi/2.0)
    Shor1.h(4)

    Shor1.h(4)
    cmod(Shor1, a)
    if C_reg[1] == 1 :
        Shor1.rz(4, pi/2.0)
    if C_reg[0] == 1 :
        Shor1.rz(4, pi/2.0)
    Shor1.h(4)

    result = run_on_device(Shor1, selected_device)

    counts = result.measurement_counts
```

Fig. 3 Fragment of the quantum circuit needed to run Shor's algorithm in Amazon Braket

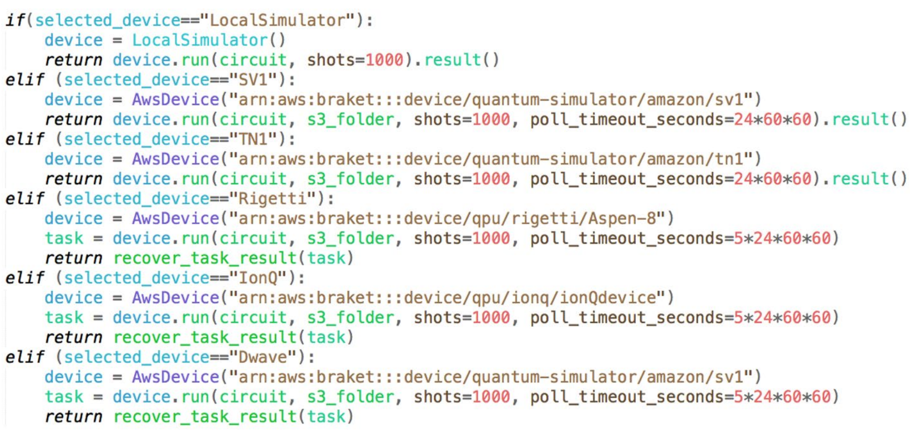
of the quantum period-finding subroutine of Shor's algorithm implemented using Amazon Braket. The complete circuit for Shor's algorithm can be executed without changes in the three simulators and the two circuit-based computers supported by Braket. Nevertheless, is interesting to note that the measurement and reinitialization of qubits supported by many other existing simulators, and that can be therefore found in public implementations of Shor's algorithm, are not supported by Braket. In the figure, this part of the algorithm is left commented as an example. Shor's algorithm can be adapted to avoid the use of these operations which means additional efforts to adapt one of the most well-known algorithms to the specifics of a given quantum platform.

This difference with other existing solutions causes that the implementation presented here only works on certain occasions. For our study, the result obtained or that the integer factorization algorithm does not work in all executions is not of great relevance because our interest is the study and analysis of the behavior of the quantum algorithm from the point of view of service-oriented computing.

Although the quantum circuit would be the same regardless of the quantum hardware or simulator used, the way in which the algorithm is invoked changes depending on where it will be run. Figure 4 shows the Braket invocation code for the three simulators and the two quantum computers supported. As can be seen in the figure, using the local simulator is the most straightforward invocation. To run the algorithms in the other simulators an s3 (Amazon simple storage system) destination has to be defined, where results will be stored, alongside a timeout for polling these results (if the polling timeout is too short, results may not be returned within the polling time). Finally, for running the algorithm on real quantum computers, a recovery task has to be defined. The quantum algorithm execution is an asynchronous operation and the developer is in charge of consulting the results when ready.

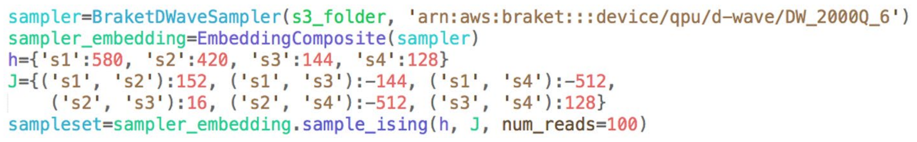
Finally, to execute integer factorization on an adiabatic quantum machine, such as D-Wave, one would have to completely rewrite the algorithm, since they are based on the adiabatic theorem closely related to quantum annealing. Thus, the mapping challenge differs from gate-based machines rendering quantum circuits inappropriate. Figure 5 shows the Braket code to factorize the number 21 using a D-Wave quantum machine.

These examples, although small, are enough to remark the current limitations of quantum software from the point of view of service-oriented computing.



```
if(selected_device=="LocalSimulator"):
    device = LocalSimulator()
    return device.run(circuit, shots=1000).result()
elif (selected_device=="SV1"):
    device = AwsDevice("arn:aws:braket:::device/quantum-simulator/amazon/sv1")
    return device.run(circuit, s3_folder, shots=1000, poll_timeout_seconds=24*60*60).result()
elif (selected_device=="TN1"):
    device = AwsDevice("arn:aws:braket:::device/quantum-simulator/amazon/tn1")
    return device.run(circuit, s3_folder, shots=1000, poll_timeout_seconds=24*60*60).result()
elif (selected_device=="Rigetti"):
    device = AwsDevice("arn:aws:braket:::device/qpu/rigetti/Aspen-8")
    task = device.run(circuit, s3_folder, shots=1000, poll_timeout_seconds=5*24*60*60)
    return recover_task_result(task)
elif (selected_device=="IonQ"):
    device = AwsDevice("arn:aws:braket:::device/qpu/ionq/ionQdevice")
    task = device.run(circuit, s3_folder, shots=1000, poll_timeout_seconds=5*24*60*60)
    return recover_task_result(task)
elif (selected_device=="Dwave"):
    device = AwsDevice("arn:aws:braket:::device/quantum-simulator/amazon/sv1")
    task = device.run(circuit, s3_folder, shots=1000, poll_timeout_seconds=5*24*60*60)
    return recover_task_result(task)
```

Fig. 4 Fragment of the Amazon Braket code to invoke the Shor's algorithm in different devices



```
sampler=BraketDWaveSampler(s3_folder, 'arn:aws:braket:::device/qpu/d-wave/DW_2000Q_6')
sampler_embedding=EmbeddingComposite(sampler)
h={'s1':580, 's2':420, 's3':144, 's4':128}
J={('s1', 's2'):152, ('s1', 's3'):-144, ('s1', 's4'):-512,
    ('s2', 's3'):16, ('s2', 's4'):-512, ('s3', 's4'):128}
sampleset=sampler_embedding.sample_ising(h, J, num_reads=100)
```

Fig. 5 Fragment of the Amazon Braket code to run the integer factorization algorithm in a D-Wave device

5 Quantum service implementation and its current limitations

After developing the described service, the algorithm was tested to execute on all the defined machines and simulators, and different metrics were used to evaluate its performance and the limitations of including a quantum service in a hybrid system. The following were evaluated: **number of qubits**, since it is one of the main limitations of current quantum computers, and this directly affects the ability to execute the service; the **number of shots**, due to the problems that arise from the characteristics of real quantum computers, mainly noise in the state of the qubits, the experiments must be performed several times or "shots" to be statistically consistent; the **precision of results**, since there is some discrepancy in the results obtained on different machines; the **response times**, this measure corresponds to the time elapsed between sending the request and receiving the result; and the **economic cost** of invoking each solution.

To proceed with the evaluation, several HTTP requests were made from the Postman API client tool[10], which allows making requests to REST APIs and taking the described metrics.

The analysis carried out during and after the experiments allows us to conclude that there is some roughness, limitations, and problems that arise when a quantum piece of software is expected to be provided as a service. The mentioned limitations are not related to the fact that quantum services cannot be built but to the fact that, by implementing quantum services with current service technologies, the potential benefits of Service-Oriented Computing are lost.

For the case study proposed above in Sect. 4, the prime number factorization service, it is not viable to make use of traditional algorithms, but it is possible to take advantage of the benefits offered by quantum computing.

In traditional service-oriented computing, replacing a service with another, even if it is deployed on different hardware architecture, is mostly trivial (at least mostly trivial if all the infrastructure is in place such as the Fig. 1 shows). However, what happens if there is a need to replace a classical service with a quantum service?

Following a similar approach to that described in the definition of classical services, an attempt will be made to replicate the classical architecture by transferring it to the quantum world in order to maintain the quality attributes provided by service-oriented computing. Each aspect of a hybrid classical-quantum architecture will then be analyzed.

Taking into account the early stage of quantum software engineering, as shown in Fig. 6, most of the boxes are empty. Regarding the **implementation layer** of a quantum service, the service business logic must be implemented, as we did for the classic service. But then,

[10] https://www.postman.com/

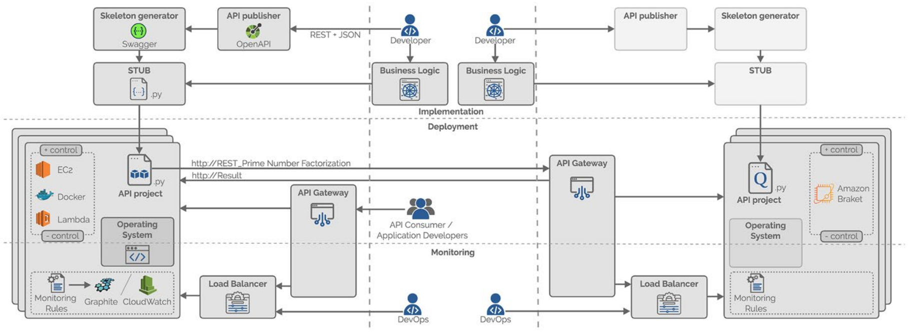


Fig. 6 A hybrid classical-quantum architecture version of a implementation of a service

we have no support or standardization mechanism for the definition of APIs for quantum endpoints. Similarly, there are no tools for code generation at the quantum service level.

In this layer, following a traditional implementation, services must be platform independent. To achieve this, a REST API (based on HTTP, IP and all other well-known Internet protocols) is used to ensure that the services can talk to each other. Similarly, JSON is used as a way to transfer data between services regardless of implementation languages. The use of the OpenAPI specification also helps achieve this independence by providing a way to define a well-formed API that can be easily consumed. However, for a quantum implementation, no communication protocols have been defined, no formatted ways exist for communicating classical services with quantum services, and no specifications even exist for defining an API to achieve service independence. Although there are already researchers who have begun to consider these issues, such as Cuomo et al. (2020) who give an overview of the main challenges and open problems that arise in the design of a distributed quantum computing ecosystem from the perspective of communications engineering; or Rojo et al. (2021) who address the challenge of giving an implementation in the form of a quantum microservice to a well-known problem such as the traveling salesman problem.

Another important feature of service-oriented computing is decoupling. At the implementation layer, the decoupling must be between the services and the programming language in which they are written. In the development of a classic service, this is provided by the OpenAPI specification and code generation tool such as Swagger that are able to generate the Stub of the service in different programming languages. However, in designing a quantum service there are no specifications and code generation tools to generate the stub structure of an API project. Although some work is found such as Dreher and Ramasami (2019) in which they have developed a prototype container-based system that allows a developer to prototype, test and implement quantum algorithms with greater agility and flexibility.

The situation does not improve in the **deployment layer**. At the moment, we have some commercial platforms in which we can execute quantum algorithms. For example, Amazon Braket allow us to run quantum algorithms in different simulators or quantum processors. However, Braket does not offer any control over the platform in which the quantum services will be executed (only one quantum task can be placed in the queue of a quantum processor and wait for a response). Similarly, the API Gateway is not prepared to deal with quantum services.

At this layer, it is also important that services are independent of the location where they are deployed. And from a classic service design perspective, this independence can be considered at two levels. At the physical level, location independence is provided by REST, HTTP, and DNS, which allow us to use a URL to access a service regardless of where it is deployed. At the logical level, location independence for a given service within a more complex project is provided by the API gateway that hides this complexity from clients. But from the point of view of a quantum service, there is no concrete definition, although there are already some works that address these aspects, as in Kumara et al. (2021) that present a vision of Quantum Service-Oriented Computing (QSOC), being a model to build hybrid business applications by placing in collaboration developers of classical services and developers of quantum services; or as Garcia-Alonso et al. (2021) that propose a Quantum API Gateway following the traditional API Gateway pattern and adapting it to the quantum world. Moreover, this Quantum API Gateway recommends the best quantum computer to execute a given quantum service at runtime.

Continuing at the deployment layer, another important feature of service-oriented computing is decoupling. From a classical service design perspective, two different types of decoupling are considered. First, decoupling between different services of the same API. This is provided by the API gateway, which can communicate the services without the services being aware of each other. And second, decoupling between services and hardware and operating systems. This is provided by the platform on which the services are deployed (EC2, Docker, Lambda, etc.). But from the point of view of a quantum service, there is no concrete definition on these decoupling aspects, but there is some work starting to work on this path, as in Grossi et al. (2021) who describe an architectural framework that addresses the problems of integrating an API-exposed quantum provider into an existing enterprise architecture and provides a minimum viable product (MVP) solution that actually merges classical quantum computers into a basic scenario with reusable code in a GitHub repository.

Finally, at the **monitoring layer**, the situation is even worse. In classical service monitoring, there are countless tools, for example *Graphite* or *CloudWatch*; however, there is no support in current tools for monitoring quantum services. The development of this type of tools can offer great benefits to obtain detailed information on the state of quantum services.

In essence, services should be resilient and able to scale to address client demand. The most common way to provide this scalability, without incurring high economic costs on the part of the cloud provider, is the use of elastic platforms and a load balancer that is capable of starting and stopping new instances of the service as needed. But the technology to develop this feature in the quantum world does not yet exist. There are incipient works in this line, such as Sete et al. (2016) in which they describe a functional architecture based on a planar lattice of qubits that allows experimental tests of quantum error correction schemes.

Services should also be composable, so that a set of simple services can be composed to provide a solution to a complex problem. In traditional service development, this composability can be provided by the API Gateway, which can decompose a single invocation into a call to different services without the services knowing about it. However, there is no formal definition for the composability of quantum services. Although there are already different works in this line, such as Wild et al. (2020) in which they introduce two deployment modeling styles based on the Topology and Orchestration Specification for Cloud Applications (TOSCA) standard to automate the deployment and orchestration of quantum applications; or such as Barzen et al. (2020) in which they present a collaborative platform for problem solving with quantum computers; or as Cohen et al. (2020) in which they define a platform for designing quantum control protocols for a wide range of quantum hardware and define how to optimize their performance.

In addition, services must also be reliable, including security, maintainability, accountability, and many other capabilities. In classical services, there is a complete set of monitoring and analysis tools that provide us with all the necessary information about the services. However, in quantum services, there is no progress in this aspect. There is a nascent work, in which researcher You (2020) proposes a framework based on quantum computation for reliability assessment of complex systems.

6 Conclusion and future work

In this paper, and based on what we already know about classical service-oriented computing, we have presented an analysis of current quantum software from the point of view of service-oriented computing. We have used Amazon Braket to deploy quantum services by wrapping them on a classical service and used the integer factorization problem to show the differences of running the same service on different quantum hardware, even when doing it under the common umbrella of Braket.

We have presented a possible implementation of a classical service following the best practices, design patterns, and existing standards. Taking this implementation as a reference, we have developed a replica using a quantum services. This has allowed us to clearly present the current limitations, the proposals that are emerging and the possibilities that we have for the development of quantum services. Given the existing limitations, we have argued that intensive research efforts are needed to bring the benefits of service-oriented computing to the quantum world.

Due to the young nature of quantum software engineering, most areas in this discipline, including service-oriented computing, are still giving their first steps. Nevertheless, the paradigm change that underlies quantum computing implies that there cannot be a direct translation of proposals and techniques. Running quantum algorithms as traditional services is not enough to bring the benefit of service-oriented computing to the quantum era. There needs to be an effort to generate new techniques, methodologies, and tools that bring all these benefits, already shown by cloud and service computing, to quantum software and services.

Acknowledgements This work was supported by the projects 0499_4IE_PLUS_4_E (Interreg V-A España-Portugal 2014-2020) and RTI2018-094591-B-I00 (MCIU/AEI/FEDER, UE), by the FPU19/03965 grant, by the Department of Economy and Infrastructure of the Government of Extremadura (GR18112, IB18030), and by the European Regional Development Fund.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Declarations

Conflict of interest The authors declare that there is no conflict of interest.

References

Barzen, J., Leymann, F., Falkenthal, M., Vietz, D., Weder, B., & Wild, K. (2020). Relevance of near-term quantum computing in the cloud: A humanities perspective. In: D. Ferguson, C. Pahl, M. Helfert (Eds.), *Cloud Computing and Services Science - 10th International Conference, CLOSER 2020* (vol. 1399, pp. 25–58). Prague, Czech Republic. Revised Selected Papers. Communications in Computer and Information Science. Springer. https://doi.org/10.1007/978-3-030-72369-9_2

Barzen, J., Leymann, F., Falkenthal, M., Vietz, D., Weder, B., & Wild, K. (2020). Relevance of near-term quantum computing in the cloud: a humanities perspective. *Communications in Computer and Information Science*, *1399 CCIS*, 25–58. https://doi.org/10.1007/978-3-030-72369-9_2

Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., Alam, M. S., Ahmed, S., Arrazola, J. M., Blank, C., Delgado, A., & Jahangiri, S., et al. (2018). Pennylane: Automatic differentiation of hybrid quantum-classical computations. arXiv preprint: arXiv:1811.04968.

Boixo, S., Albash, T., Spedalieri, F. M., Chancellor, N., & Lidar, D. A. (2013). Experimental signature of programmable quantum annealing. *Nature Communications, 4*(1), 1–8.

Bouguettaya, A., Singh, M., Huhns, M., Sheng, Q. Z., Dong, H., Yu, Q., et al. (2017). A service computing manifesto: the next 10 years. *Communications of the ACM, 60*(4), 64–72.

Cheng, J. K., Lim, E. M., Krikorian, Y. Y., Sklar, D. J., & Kong, V. J. (2021). A survey of encryption standard and potential impact due to quantum computing. *IEEE Aerospace Conference Proceedings*. https://doi.org/10.1109/AERO50100.2021.9438392

Cohen, Y., Sivan, I., Ofek, N., Ella, L., Drucker, N., Shani, T., Weber, O., Grinberg, H., & Greenbaum, M. (2020). Quantum orchestration platform integrated hardware and software for design and execution of complex quantum control protocols. *Bulletin of the American Physical Society*, *65*(1).

Cuomo, D., Caleffi, M., & Cacciapuoti, A. S. (2020). Towards a distributed quantum computing ecosystem. *IET Quantum Communication, 1*(1), 3–8. https://doi.org/10.1049/IET-QTC.2020.0002

Digital Journal. (2022). Topological quantum computing market is likely to experience a tremendous growth in near future. https://www.digitaljournal.com/pr/topological-quantum-computing-market-is-likely-to-experience-a-tremendous-growth-in-near-future-microsoft-ibm-google-d-wave-systems

Dreher, P., & Ramasami, M. (2019). Prototype container-based platform for extreme quantum computing algorithm development. *2019 IEEE High Performance Extreme Computing Conference*. HPEC 2019. https://doi.org/10.1109/HPEC.2019.8916430

Endrei, M., Ang, J., Arsanjani, A., Chua, S., Comte, P., Krogdahl, P., Luo, M., & Newling, T. (2004). *Patterns: Service-oriented architecture and web services*. IBM Corporation, International Technical Support Organization New York, NY.

Garcia-Alonso, J., Rojo, J., Valencia, D., Moguel, E., Berrocal, J., & Murillo, J. M. (2021). Quantum software as a service through a quantum API gateway. *IEEE Internet Computing*.

Grossi, M., Crippa, L., Aita, A., Bartoli, G., Sammarco, V., Picca, E., Said, N., Tramonto, F., & Mattei, F. (2021). *A serverless cloud integration for quantum computing*. arXiv:2107.02007.

Haring, R., Ohmacht, M., Fox, T., Gschwind, M., Satterfield, D., Sugavanam, K., et al. (2011). The IBM blue gene/q compute chip. *IEEE Micro, 32*(2), 48–60.

Haugeland, S. G., Nguyen, P. H., Song, H., & Chauvel, F. (2021). Migrating monoliths to microservices-based customizable multi-tenant cloud-native apps. *Proceedings - 2021 47th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2021* (pp. 170–177). https://doi.org/10.1109/SEAA53835.2021.00030

Hu, F., Wang, B. -N., Wang, N., & Wang, C. (2019). Quantum machine learning with d-wave quantum computer. *Quantum Engineering, 1*(2), 12.

Hooton, C. (2019). Examining the economic contributions of the cloud to the united states economy. Report. Internet Association. Washington, DC.

Jiang, S., Britt, K. A., McCaskey, A. J., Humble, T. S., & Kais, S. (2018). Quantum annealing for prime factorization. *Scientific Reports, 8*(1), 1–9.

Kielpinski, D., Monroe, C., & Wineland, D. J. (2002). Architecture for a large-scale ion-trap quantum computer. *Nature, 417*(6890), 709–711.

Kumara, I., Heuvel, W. -J. V. D., & Tamburri, D. A. (2021). QSOC: Quantum service-oriented computing. In: *Symposium and Summer School on Service-Oriented Computing* (pp. 52–63). Springer. https://doi.org/10.1007/978-3-030-87568-8_3, https://link.springer.com/chapter/10.1007/978-3-030-87568-8_3

MacQuarrie, E. R., Simon, C., Simmons, S., & Maine, E. (2020). The emerging commercial landscape of quantum computing. *Nature Reviews Physics, 2*(11), 596–598.

Mazlami, G., Cito, J., & Leitner, P. (2017). Extraction of microservices from monolithic software architectures. *Proceedings - 2017 IEEE 24th International Conference on Web Services, ICWS 2017* (pp. 524–531). https://doi.org/10.1109/ICWS.2017.61

McCaskey, A. J., Lyakh, D. I., Dumitrescu, E. F., Powers, S. S., & Humble, T. S. (2020). Xacc: a system-level software infrastructure for heterogeneous quantum-classical computing. *Quantum Science and Technology, 5*(2), 024002.

McCaskey, A., Dumitrescu, E., Liakh, D., & Humble, T. (2018). Hybrid programming for near-term quantum computing systems. In: *2018 IEEE International Conference on Rebooting Computing (ICRC)* (pp. 1–12). IEEE.

Moguel, E., Berrocal, J., García-Alonso, J., & Murillo, J. M. (2020). A roadmap for quantum software engineering: Applying the lessons learned from the classics. In: R. Pérez-Castillo, M. Piattini, G. Peterssen, J. L.Hevia (Eds.), *Short Papers Proceedings of the 1st International Workshop on Software Engineering & Technology (Q-SET'20) Co-located with IEEE International Conference on Quantum Computing and Engineering (IEEE Quantum Week 2020), (Online Conference)* (vol. 2705, pp. 5–13). Broomfield, Colorado, USA. CEUR Workshop Proceedings. CEUR-WS.org. http://ceur-ws.org/Vol-2705/short1.pdf

Motta, M., Sun, C., Tan, A. T., O'Rourke, M. J., Ye, E., Minnich, A. J., et al. (2020). Determining eigenstates and thermal states on a quantum computer using quantum imaginary time evolution. *Nature Physics, 16*(2), 205–210.

Newman, S. (2021). Building microservices. Oreilly.

Nielsen, M. A., & Chuang, I. (2002). Quantum computation and quantum information. American Association of Physics Teachers.

Papazoglou, M. P. (2003). Service-oriented computing: Concepts, characteristics and directions. In: *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, 2003 (pp. 3–12). WISE 2003. IEEE.

Pérez-Castillo, R., Serrano, M. A., & Piattini, M. (2021). Software modernization to embrace quantum technology. *Advances in Engineering Software, 151,* 102933.

Pérez-Castillo, R., & Piattini, M. (2020). The quantum software engineering path. In: R. Pérez-Castillo, M. Piattini, G. Peterssen, J. L. Hevia (Eds.), *Short Papers Proceedings of the 1st International Workshop on Software Engineering & Technology (Q-SET'20) Co-located with IEEE International Conference on Quantum Computing and Engineering (IEEE Quantum Week 2020)* (vol. 2705, pp. 1–4). Broomfield, Colorado, USA. CEUR Workshop Proceedings. CEUR-WS.org. http://ceur-ws.org/Vol-2705/invited1.pdf

Piattini, M., Peterssen, G., & Pérez-Castillo, R. (2020). Quantum computing: a new software engineering golden age. *ACM SIGSOFT Softw Eng Notes, 45*(3), 12–14. https://doi.org/10.1145/3402127.3402131

Piattini, M., Serrano, M., Perez-Castillo, R., Petersen, G., & Hevia, J. L. (2021). Toward a quantum software engineering. *IT Professional, 23*(1), 62–66.

Piattini, M., Peterssen, G., Pérez-Castillo, R., Hevia, J. L., Serrano, M. A., Hernández, G., de Guzmán, I. G. R., Paradela, C. A., Polo, M., Murina, E., Jiménez, L., Marqueño, J. C., Gallego, R., Tura, J., Phillipson, F., Murillo, J. M., Niño, A., & Rodríguez, M. (2020). The talavera manifesto for quantum software engineering and programming. In: *Short Papers Proceedings of the 1st International Workshop on the Quantum Software Engineering & Programming* (vol. 2561, pp. 1–5). Talavera de la Reina, Spain. CEUR Workshop Proceedings.

Pistoia, M., Ahmad, S. F., Ajagekar, A., Buts, A., Chakrabarti, S., Herman, D., Hu, S., Jena, A., Minssen, P., Niroula, P., Rattew, A., Sun, Y., & Yalovetzky, R. (2021). Quantum machine learning for finance. arXiv:2109.04298. https://doi.org/10.1109/ICCAD51958.2021.9643469

Rahaman, M., & Islam, M. M. (2015). A review on progress and problems of quantum computing as a service (GCAAS) in the perspective of cloud computing. *Global Journal of Computer Science and Technology.*

Ravichandran, T., & Rai, A. (2000). Quality management in systems development: an organizational system perspective. *MIS Quarterly: Management Information Systems, 24*(3), 381–410. https://doi.org/10.2307/3250967

Richardson, C. (2019). *Microservices Patterns* (p. 490). Manning Publications.
Rojo, J., Valencia, D., Berrocal, J., Moguel, E., Garcia-Alonso, J., & Rodriguez, J. M. M. (2021). *Trials and tribulations of developing hybrid quantum-classical microservices systems*. arXiv:2105.04421
Sete, E. A., Zeng, W. J., & Rigetti, C. T. (2016). A functional architecture for scalable quantum computing. *2016 IEEE International Conference on Rebooting Computing*. ICRC 2016 - Conference Proceedings. https://doi.org/10.1109/ICRC.2016.7738703
Sodhi, B. (2018). Quality attributes on quantum computing platforms. arXiv preprint: arXiv:1803.07407
Wang, B., Hu, F., Yao, H., & Wang, C. (2020). Prime factorization algorithm based on parameter optimization of Ising model. *Scientific Reports, 10*(1), 1–10.
Wang, L., Von Laszewski, G., Younge, A., He, X., Kunze, M., Tao, J., & Fu, C. (2010). Cloud computing: a perspective study. *New Generation Computing, 28*(2), 137–146.
Wild, K., Breitenbucher, U., Harzenetter, L., Leymann, F., Vietz, D., & Zimmermann, M. (2020). TOSCA4QC: Two modeling styles for TOSCA to automate the deployment and orchestration of quantum applications. *Proceedings - 2020 IEEE 24th International Enterprise Distributed Object Computing Conference* (pp. 125–134). EDOC 2020. https://doi.org/10.1109/EDOC49727.2020.00024
Wille, R., Van Meter, R., & Naveh, Y. (2019). IBM's qiskit tool chain: Working with and developing for real quantum computers. In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 1234–1240). IEEE.
Wolff, E. (2019). *Microservices - a practical guide: Principles, concepts, and recipes*. Lean Publishing.
You, S. (2020). A quantum computing framework for complex system reliability assessment. arXiv:2012.03919.
Zhao, J. (2020). Quantum software engineering: Landscapes and horizons. CoRR abs/2007.07047. arXiv:2007.07047
Zinner, M., Dahlhausen, F., Boehme, P., Ehlers, J., Bieske, L., & Fehring, L. (2022). *Drug Discovery Today, 27*(2), 378–383. https://doi.org/10.1016/J.DRUDIS.2021.10.006

Enrique Moguel is an assistant professor at the University of Extremadura (Spain). He completed his MSc in Computer Science at the University Carlos III (Spain) in 2010 and a PhD in Computer Science at the University of Extremadura in 2018. His research interests include Web Engineering, Smart Systems, and Quantum Computing.

Javier Rojo is a PhD student at the University of Extremadura. He has a predoctoral position at the same university, through a grant for the formation of university professors, from the Spanish Ministry of Science, Innovation and Universities. He completed his MSc. in Computer Science at the University of Extremadura in 2021. His research interests include software engineering, eHealth, pervasive computing, and mobile computing.

David Valencia is a postdoctoral researcher at the University of Extremadura (Spain). He completed his MSc (2004) and PhD (2010) in Computer Science at the University of Extremadura (Spain). His research interests include Quantum Computing, Parallel and Distributed Computing, and Software Engineering.

Javier Berrocal is an associate professor in the Department of Informatics and Telematics System Engineering at the University of Extremadura (Spain) and co-founder of the Startup Gloin. He completed his PhD degree (with European Mention) in 2014. His research interests include mobile computing, context awareness, pervasive systems, the Internet of Things, and fog computing.

Jose Garcia-Alonso is an associate professor in the Department of Informatics and Telematics System Engineering at the University of Extremadura (Spain) and co-founder of the Startups Gloin, Viable and Health and Ageing Tech. He completed his PhD degree (with European Mention) in 2014. His main research interests are in Quantum Software Engineering, eHealthCare, eldercare, Mobile Computing, Context-Awareness, and Pervasive Systems.

Juan M. Murillo is a full professor of software engineering at the University of Extremadura and co-founder of the Startups Gloin and Viable. His research interests include Quantum Software Engineering, software architectures, mobile computing, and cloud computing.

Authors and Affiliations

Enrique Moguel[1] · Javier Rojo[1] · David Valencia[1] · Javier Berrocal[1] · Jose Garcia-Alonso[1] · Juan M. Murillo[1]

Javier Rojo
javirojo@unex.es

David Valencia
davaleco@unex.es

Javier Berrocal
jberolm@unex.es

Jose Garcia-Alonso
jgaralo@unex.es

Juan M. Murillo
juanmamu@unex.es

[1] Quercus Software Engineering Group, Universidad de Extremadura, Avda. de la Universidad, s/n, Cáceres 10004, Spain