1 Requirements for an Ontology Supporting Quantum Computing Adoption

This chapter aims to specify the ontology's requirements following the methodology proposed in Section 3.2.1. First, the user stories, represented by the outcomes of the expert interviews (Chapter 4), are complemented with a review of related work to identify functional and non-functional requirements (tasks 4-6 of Figure 12). Subsequently, these requirements are grouped to support the modularity of the ontology design and then translated into competency questions. As a result, this ontology requirement specification contributes to answering SRQ 5.

Section 5.1 denotes the functional requirements that specify the knowledge represented by the ontology, based on the inhibitors and enablers of quantum computing adoption found in Section 4.1 and Section 4.2. Next, Section 5.2 briefly describes the non-functional requirements of the ontology design, specifically focused on mitigating the implications stated in Section 4.3. Lastly, competency questions are composed in Section 5.3, representing the tasks the ontology should cover.

1.1 Functional Requirements

As above-mentioned, functional requirements refer to knowledge represented by the ontology. This knowledge is specified by reviewing relevant literature and applications aimed at mitigating and stimulating the inhibitors and enablers of quantum computing adoption, respectively.

Firstly, Section 5.1.1 defines the knowledge required to represent a use case accurately, thereby directly contributing to the second enabler of quantum computing adoption (Table 6). Furthermore, by identifying and storing use cases, the ontology is expected to alleviate inhibitors 1, 6, 7, and 8 of quantum computing adoption (Table 5). Secondly, Section 5.1.2 denotes instances that facilitate knowledge sharing amongst organizations, which relates to the first enabler of quantum computing adoption (Table 6).

Publications referenced throughout Section 5.1.1 and Section 5.1.2 are listed in Table 8 to support brief citations.

Table 1, References used throughout knowledge specification

#	Reference
1	(Martyniuk et al., 2021)
2	(PlanQK, 2021)
3	(Leymann et al., 2020)
4	(Vietz et al., 2021)
5	(Moussa et al., 2020)
6	(Aaronson, 2015)
7	(Leymann & Barzen, 2020)
8	(Salm et al., 2020)
9	(Montanaro, 2016)
10	(Jordan, 2021)
11	(Quantum-Inspire, n.d.)
12	(Larose, 2019)
13	(Kircz & de Waard, 2003)

1.1.1 Use Case Specification

Business Problem Definition

As can be deducted from the use case diagram illustrated in Figure 13, the description of the business problem and its mapping to a (computational) problem type initiates the quantum computing application lifecycle. However, most expert interviewees indicated that non-experts often fail to classify their business problems sufficiently, as mapping the business' computational problem is perceived as complex (I6 and I7, Table 5).

Table 9 provides an overview of the knowledge specified to support defining these problems. Firstly, the ontology should represent (computational) problem types in hierarchical decomposition. By providing the hierarchy of problem types (e.g., mathematical → optimization → combinatorial optimization →traveling salesman problem), non-experts are guided in mapping their business problem. Furthermore, these problems relate to complexity classes that group problems by difficulty. These classes range from P, defining "can be solved by a deterministic classical computer in polynomial time," to QMA, defining "solution can be checked by a quantum computer in polynomial time" (Montanaro, 2016, p.2). Lastly, in the current noisy intermediate-scale quantum (NISQ) era, quantum computers are limited due to erroneous (i.e., noisy) qubits, which prevents limits to the number of qubits on a functioning quantum computer (Leymann & Barzen, 2020). So, in line with the progression of quantum computers, quantum algorithms will progress and explore potential problem areas (Montanaro, 2016). Therefore, keeping track of application areas for quantum algorithms is valuable.

Table 2, Knowledge specification for problem definitions

Classes	Sub-classes	Examples	References
Problem type	Artificial Intelligence	Computer vision, scheduling, NLP	[2], [1], [3]
	Machine Learning		[2], [1], [3]
	Mathematical	Optimization, linear algebra	[2], [1], [3]
	Search	Heuristic, randomized, discrete	[2], [1], [3]
	Sorting		[2], [1], [3]
Complexity class		P, BPP, BQP, NP, QMA	[1], [9]
Application area		Cryptography, simulation, linear equations	[2],[1], [3], [9], [6]

Quantum Algorithms

Subsequently, these computational problems are solved by an appropriate algorithmic (quantum) solution. Quantum algorithms are described by circuits structured by collections of quantum gates. Due to the complexity of capturing these circuits in a knowledge base, Leymann et al. (2020) propose handling quantum algorithms as artifacts. In this way, the algorithms are explained without considering the technology that established these artifacts. Correspondingly, the outcomes of the expert interviews conclude that experts prefer communicating at a lower level of abstraction (I3, Table 5).

To begin with, Table 10 denotes the classifications of algorithms. Firstly, quantum algorithms can be distinguished into multiple categories (i.e., algorithm classification), such as algebraic and number-theoretic algorithms, approximation and simulation algorithms, and oracular algorithms (Jordan, 2021). Furthermore, quantum algorithms can achieve a speedup over the best-performing classical algorithm, and this speedup is defined with polynomial factors (Rønnow et al., 2014). For example, a polynomial function could be T (e.g., quantum takes 100 steps) over T^2 (e.g., classical takes 10,000 steps), which means the quantum algorithm achieves polynomial speedup.

In addition, quantum algorithms express computational models, which can be seen as mathematical rules defining how quantum computation is performed (Vietz et al., 2021). These models can be categorized as universal quantum computation and restricted quantum computation.

Table 3, Knowledge specification for quantum algorithms

Classes	Sub-classes	Examples	References
Algorithm classification		Algebraic and number-theoretic algorithms, optimization	[1], [10]
Algorithm description			[3], [10]
Computational model	Universal, restricted	Gate-based model	[2], [1], [4],
Speedup Class		Super polynomial	[10]

Software in Implementation

The implementation of a quantum algorithm is centered on the interviewed experts' definition of a use case. The second enabler for quantum computing adoption (Section 4.2) discusses the importance of compatibility between software and hardware components of an implementation. The following software-related concepts are relevant to the implementation of quantum algorithms.

Firstly, access methods provide access to a computational quantum resource. Naturally, these methods are hardware-dependent, which demands compatibility between the method and the resource (Vietz et al., 2021). Three access methods are web services (e.g., IBMQ's rest API), graphical user interfaces, or command-line interfaces (e.g., AWS CLI).

Furthermore, software development kits (SDKs) are collections of advanced developer tools. Some SDKs can also access computational quantum resources, such as Forest and Qiskit. Three types of software typically contained in SDKs are local simulators (e.g., PyLinalg in byMyQLM), compilers (e.g., Quilc in Forest), and libraries (Vietz et al., 2021). Although, software contained in an SDK can also be considered a standalone technology (e.g., ScaffCC).

An implementation uses programming languages often based on the computational quantum resource. This research distinguishes four types of programming languages: assembly, quantum, integration, and syntax (Larose, 2019; Vietz et al., 2021). Assembly languages (e.g., Quil) are so-called instruction languages that provide instructions to the computational resource on what operations should be performed on the qubits. Quantum programming languages (e.g., Quipper) are higher in level than assembly languages and provide features such as loops and recursion. Integration languages (e.g., Orquestra), or workflow languages, organize the workflow between classical and quantum resources in hybrid implementations. Syntax programming languages host these programming languages, such as Python for pyQuil.

Table 4, Knowledge specification for implementational software

Classes	Sub-classes	Examples	Reference(s)
Access methods	Command-line interface	AWS CLI	[4] [1]
	Software development kits	Forest, Qiskit, Ocean	[4] [1]
	Graphical user interface		[4] [1]
	Web services	IBMQ rest-API	[4] [1]

Programming languages	Assembly	Quil, QWIRE	[1], [3], [4], [12]
	Quantum programming	Quipper, Q#	[1], [3], [4], [12]
	Integration	Orquestra	[1], [3], [4]. [12]
	Syntax	Python	[1], [3], [4]. [12]
Software development kits	Libraries	Grove, TF Quantum	[4], [1]
	Compilers & Transpilers	OpenQL, ScaffCC	[4]
	Local simulator	QuEST, PyLinalg	[4]
Software properties	Version		[12], [2]
	License	Apache-2.0	[12], [2]
	Operating system	Linux	[12]
	Provider/Contributor		[2]

Hardware in Implementation

Similarly to the software, as mentioned earlier, hardware-related knowledge is defined to complement the software concepts, thereby representing an algorithm's implementation.

Within this domain, it is identified that the major hardware involved in implementation is the computational resource. A Quantum processing unit (QPU) is the physical hardware that can perform quantum computations. For example, the IBMQX5 16-qubit QPU by resource provider IBM Quantum services, or Rigetti's Aspen-10 QPU, managing 32 qubits. However, in the NISQ-era, quantum computers are limited, making running substantial computations impossible. Alternatively, quantum simulators (e.g., in the cloud) simulate quantum computation on classical hardware. Furthermore, the qubit technologies enabling the implementation of a QPU are defined, such as photon polarization or spin states of atoms and electrons (Quantum-Inspire, n.d.).

Furthermore, the quantum computational resource is characterized by its resource properties, as seen in Table 12. For example, the gate set (i.e., circuit depth) denotes the number of gates the resource can perform sequentially, while the circuit width denotes the number of qubits available to manipulate (Leymann & Barzen, 2020). Furthermore, as previously stated, qubits are noisy and erroneous. Accordingly, coherence duration refers to the quality of qubits (i.e., time without errors). Similarly, gate fidelity defines the quality of the gates, measured by the average distance between gates (Leymann & Barzen, 2020). Lastly, the qubit connectivity refers to the topology of qubits, which optimally is all-to-all (Larose, 2019).

Table 5, Knowledge specification for implementational hardware

Classes	Sub-classes	Examples	Reference(s)
Computational resource type	Quantum Processing Unit	IBMQX5, Acorn	[2], [1], [4], [12]
	Quantum simulator	Local, Cloud	[1], [4]
	Technology	Polarization of photon, Nuclear spin states of atom	[2], [11], [8], [1]
Quantum resource properties		Gate set / Circuit depth	[2], [1], [3], [7], [8]
		Qubit connectivity	[2], [1], [12]

	Number qubits / Circuit width	[2], [1], [3], [7], [8]
	Gate fidelity (2-Qubit Gate, Readout, 1-Qubit Gate)	[2], [1], [7]
	qRAM	[2], [1]
	Coherence times	[5], [12], [7]
Resource provider	Rigetti, IBMQ	[2], [5], [3], [8], [1], [12]

Context-Dependent Execution of an Implementation

Lastly, implementations can be executed by various organizations in different contexts. Therefore, the input and output of executions are captured to express this contextuality. A description of input data is required as quantum hardware's limitations apply to quantum algorithms and data processing. Therefore, the desired input determines the type of implementation of a quantum algorithm (Salm et al., 2020). Furthermore, the performance (e.g., runtime, space usage) of an implementation for a particular organization is also captured. Containing performance knowledge in the ontology constitutes the last step of E2 (i.e., Table 6), thereby covering the use case diagram in Figure 13. Furthermore, gathering performance data about quantum computing applications mitigates the third implication of a quantum divide (i.e., Table 7) by creating increased awareness and certainty of the algorithm's performances.

Table 6, Knowledge specification for executions of an implementation

Classes	Examples	References
Organization		-
Input	Data, Parameters	[2], [1], [8] , [5]
Output	Performance measures (e.g., runtime, space usage)	[2], [1], [8], [3], [5]

1.1.2 Enabling Knowledge Sharing

According to the expert interviews (i.e., E1, Table 6), organizations are willing to share knowledge about quantum computing, which benefits the creation of use cases and creates a realistic view of the state of development in the quantum computing domain. To facilitate knowledge sharing amongst practicing organizations, executive organizations, and software and hardware providers are readily stated in Table 13, Table 11, and Table 12, respectively.

Chapter 1 addressed the immense amount of scientific work published yearly, to which quantum computing increasingly contributes. However, as stated in Section 4.1, depth in these publications is often lacking, making them abstract. Therefore, scientific publications are related to algorithms, implementations, or particular executions to ease searching for valuable and high-quality publications. Kircz & de Waard (2003) identified relevant metadata for science publishing, of which the most important ones are stated in Table 14.

Table 7, Knowledge specification of publication metadata

Examples	References
Authors	[2], [1], [13]
Title	[2], [13]
Link (DOI or URL)	[2], [1], [13]
Abstract	[1], [13]
Date	[1], [13]

1.2 Non-Functional Requirements

The ontology design's non-functional requirements aim to mitigate the implications of the quantum divide. These implications, stated in Table 7 in Section 4.3, are considered ethically significant, as they potentially affect the chances of having a good life on an individual level or flourishing on a societal level—for example, the outcomes of irresponsible research and innovation or monopolization of organizations, respectively.

Based on the guidance ethics approach by Verbeek & Tijink (2020), ethics by design is considered in the definition of non-functional solution requirements. With this approach, explicit ethical values are incorporated into the solution. The following four non-functional requirements are based on ethical values for quantum technologies recommended by Coenen et al. (2022).

Comprehensibility

Coenen et al. define a comprehensible presentation of quantum technologies as "legible, honest, and publicly accountable." (2022, p. 5). A comprehensible presentation alleviates the fifth inhibitor of Table 5 (Section 4.1), which concerns pursuing an adaptive communication strategy to communicate quantum knowledge to multiple strata of the society effectively.

Ławrynowicz et al. (2016) propose a three-layered ontology structure to support different uses of their machine learning knowledge base, as they distinguish between specifications, implementations, and executions. This research already pursues a modular ontology design, which has proven to facilitate reusability and maintainability (Scrocca et al., 2021). In incorporating the usability of different end-users in decisions regarding modularity, this research distinguishes three "layers": applications (i.e., non-expert adopters), algorithms (i.e., experts), and implementations (i.e., adopter-expert collaboration).

Specificity

Moreover, Coenen et al. state that quantum technologies should be attended to with "specific innovation pathways, socio-technical processes and designs, denoting new resources, application fields, relevant actors, and development strategies" (2022, p. 5). By ensuring specificity, the knowledge base gathers and defines accurate representations of use cases, enabling the second enabler of Table 6 in Section 4.2. The functional requirements in Section 5.1.1 specify the knowledge representing such resources, application fields, and actors.

Openness

Thirdly, research on quantum computing should be available to "communities beyond early adopter states, start-ups, and big tech companies" (Coenen et al., 2022, p. 5). To satisfy this requirement, the source code of modeling and implementation of the ontology is made open-source by publicizing the GitHub repository (https://github.com/julianmartens98/Master_Thesis). In addition, references to the knowledge stored in the ontology (i.e., publications metadata) are maintained to ensure transparency and enable further exploration if desired.

Accessibility

Lastly, Coenen et al. recommend accessibility " to enhance the diversity of the field's workforce, design implementations that anticipate and support the greatest variety of users and contexts" (2022, p. 5). This recommendation corresponds with the third implication of the quantum divide, as current quantum computing adopters are often large industrial concerns or banks. As stated in Section 5.1.2, the denotation of organizations involved in the quantum computing application's lifecycle is valued in

the ontology design. By specifying these organizations with, for example, their application area, similar organizations can be inspired to engage in quantum computing.

1.3 Competency Questions

Competency questions (CQ) capture the requirements of the previous chapters in a set of tasks. These questions characterize the ontology, as they define which axioms (i.e., expressions) the ontology should contain to answer these questions (Grüninger & Fox, 1995).

Business Problems

Quantum computing non-expert adopters from the industry demand exemplary problem definitions to identify their relevance in creating use cases. The following CQs support the request for problem definitions.

1. What problem is solved with an algorithm execution?
2. How can the problem be described?
3. To what problem type belongs the problem?
4. To what complexity class belongs the problem?
5. To what application area belongs a problem?

Quantum Algorithms

End-users of the knowledge base should be able to understand the conceptualization of an algorithm.

6. To what algorithm type belongs the algorithm?
7. How can the algorithm be described?
8. To what speedup class belongs the algorithm?
9. What is the computational model that expresses the algorithm?
10. How is the algorithm implemented?
11. What is the execution of the algorithm?

Software in Implementation

The implementation of a quantum algorithm requires various software. The following CQs are aimed at capturing the software used in an implementation.

12. What software is used in the implementation?
    a. What libraries are used in the implementation?
    b. What compilers and transpilers are used in the implementation?
    c. What access method is used in the implementation?
    d. What programming languages are used in the implementation?
        i. What assembly languages are used in the implementation?
        ii. What are quantum programming languages used in the implementation?
        iii. What integration languages are used in the implementation?
        iv. What are syntax languages used in the implementation?
13. What software is contained in a software development kit?
14. What properties characterize the software?
    a. What is the version of the software?
    b. What is the license of the software?
    c. What are the compatible operating systems of the software?
15. What software is compatible with the computational resource?
16. What organization provides the software?
17. What organization developed the implementation?

Hardware in Implementation

Similarly, implementing an algorithm requires hardware (i.e., a quantum computational resource).

18. What is the computational resource used in the implementation?
19. What is the technology required by the computational resource?
20. What properties characterize the computational resource?
    a. What number of qubits has the computational resource?
    b. What gate set has the computational resource?
    c. What qubit connectivity has the computational resource?
    d. What gate fidelity has the computational resource?
    e. What qRAM has the computational resource?
    f. What is the coherence time of the computational resource?
21. What organization provides the computational resource?
22. What technology is used by the computational resource?
23. What computational model is compatible with the computational resource?

Context-Dependent Execution of an Implementation

Executing an implementation is context-dependent, as the execution involves an organization with particular input and output.

24. What implementation was used in the execution?
25. What organization was involved in the execution?
26. What is the input of the execution?
    a. What was the data used in the execution?
    b. What were the parameters for the execution?
27. What output resulted from an execution?
    c. What performance was achieved by the execution?

Knowledge-sharing

Corporations are willing to work together, supporting the creation of use cases. However, end-users of the knowledge base must be able to find organizations. Thus, these organizations should be characterized. The following CQs are aimed to support this recognizability.

28. What is the classification/application area of the organization?
29. Where is an organization situated?

In addition, the knowledge base should ease the search for valuable and high-quality publications. Processing the following CQs, aimed at capturing metadata, support this search.

30. What are the publications about quantum algorithms?
31. What are the publications about implementations?
32. What are the publications about executions?
33. What metadata describes a publication?
    a. Who are the authors of a publication?
    b. What is the title of a publication?
    c. What is the abstract of a publication?
    d. What is the link to a publication?
    e. What is the date of a publication?