УНИВЕРЗИТЕТ У НОВОМ САДУ

ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Лука Радујевић

Квантно рачунарство у пракси: увид у употребљивост и кључни закључци из перспективе софтверског инжењера

МАСТЕРСКИ РАД

Мастер академске студије

Нови Сад, 2023.

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:			
Идентификациони број, **ИБР**:			
Тип документације, **ТД**:		Монографска документација	
Тип записа, **ТЗ**:		Текстуални штампани материјал	
Врста рада, **ВР**:		Завршни (Master) рад	
Аутор, **АУ**:		Лука радујевић	
Ментор, **МН**:		др Владимир Мандић	
Наслов рада, **НР**:		Квантно рачунарство у пракси: увид у употребљивост и кључни закључци из перспективе софтверског инжењера	
Језик публикације, **ЈП**:		Српски / ћирилица	
Језик извода, **ЈИ**:		Српски	
Земља публиковања, **ЗП**:		Република Србија	
Уже географско подручје, **УГП**:		Војводина	
Година, **ГО**:		2023	
Издавач, **ИЗ**:		Ауторски репринт	
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6	
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		8/64/36/3/18/0/0	
Научна област, **НО**:		ИМТ – Информационе технологије	
Научна дисциплина, **НД**:		Инжењерство информационих система	
Предметна одредница/Кqучне речи, **ПО**:		квантно рачунарство, квантно софтверско инжењерство	
УДК			
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, **ВН**:			
Извод, **ИЗ**:		Предности и изазови које квантно рачунарство уводи. Програмирање у квантном рачунарству и важни концепти који се користе. Компаративна анализа квантног софтверског инжењерства и класичног софтверског инжењерства. Изазови из угла софтверског инжењера у раду са квантним технологијама.	
Датум прихватања теме, **ДП**:			
Датум одбране, **ДО**:			
Чланови комисије, **КО**:	Председник:	др Ђорђе Пржуљ, ванредни професор	
	Члан:	др Никша Јаковљевић, ванредни професор	Потпис ментора
	Члан, ментор:	др Владимир Мандић, ванредни професор	

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Bachelor Thesis	
Author, **AU**:		Luka Radujević	
Mentor, **MN**:		Vladimir Mandić, PhD	
Title, **TI**:		Quantum Computing in Practice: Usability Insights and Key Takeaways from the Perspective of a Software Engineer	
Language of text, **LT**:		Serbian	
Language of abstract, **LA**:		Serbian	
Country of publication, **CP**:		Republic of Serbia	
Locality of publication, **LP**:		Vojvodina	
Publication year, **PY**:		2023	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		8/64/36/3/18/0/0	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S/KW**:		quantum computing, quantum softvare engineering	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		Advantages and challenges introduced by quantum computing. Programming in quantum computing and important concepts used. Comparative analysis of quantum software engineering and classical software engineering. Challenges from the perspective of a software engineer working with quantum technologies.	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	Đorđe Pržulj, PhD, associate professor	
	Member:	Nikša Jakovljević, PhD, associate professor	Menthor's sign
	Member, Mentor:	Vladimir Mandić, PhD, associate professor	

Садржај

Слике

Листинзи

Табеле

1. Увод

Развој софтверског рачунарства се кроз историју дешавао захваљујући, пре свега, напретку хардвера на ком су се програми извршавали [1]. Програмске парадигме које су коришћене током педесетих година двадестог века биле су условљене компонентама рачунара које су у том тренутку биле развијене и програми су писани тако да су у великој мери били прилагођени ограничењима капацитета. Како су процесори, радне меморије и тврди дискови добијали све веће капацитете, програмске парадигме су се мења тако да су се све мање осврталe на хардвер и све већи акценат је стављан на писање кода који би био што јаснији програмерима и који би имао могућност лаког проширења и модификације.

Поред напретка рачунарске моћи, кроз историју је постојала велика тенденција за смањивањем величине хардверских компоненти. Први рачунари су били сачињени од великих вакуумских и електронских цеви које биле великих димензија и заузимале су значајан простор. Са напретком полупроводничке технологије и појавом транзистора, настали су микропроцесори, који су омогућили знатно смањење димензија рачунара и њихову употребу у широј популацији [1]. Рачунарске компоненте су временом додатно смањене тако да су могле да стану у мобилни телефон и минијатурне сензоре. Прецизније формулисано, смањење димензија се односи на смањење физичких материјала који под дејством електромагнетног поља мењају своје особине. Када се ови материјали смање на ниво честица, њихово понашање престаје да описује класична физика и примењују се закони које описује квантна физика.

Са напретком истраживања у области квантне физике, родила се идеја и концепт који се назива квантно рачунарство (енг. *Quantum Computing*) [3][14]. Концепт квантног рачунарства уводи појмове и концепте који омогућавају фундаментално другачији приступ за решавање рачунарских проблема. Најпознатији међу појмовима које уводи квантно рачунарство су појам квантног бита, односно *кјубита* (енг. *Qubit*), потом појмова *суперпозиције* (енг. *Superposition*) и *квантног спрезања* (енг. *Entanglement*) [10]. Поменути концепти су са собом донели могућност дизајнирања недетерминистичког система, које другим речима зовемо *квантно програмирање* (енг. *Quantum Programming*). На овај начин настаје читав нови свет могућности које би могле да реше многе проблеме које до тад познато, класично рачунарство, није могло да досегне. Један од таквих проблема јесу симулације квантних система и симулације хемијских реакција, где се помоћу квантног рачунарства репрезентују молекуларне структуре и симулирају њихове интеракције, много ефикасније него што се то могуће коришћењем класичног рачунарства [3]. Поред симулација, додатни проблеми који могу ефикасније да се реше коришћењем квантног рачунарства су факторизација великих бројева, машинско учење, вештачка интелигенција и криптографија [29]. У контексту криптографије, квантно рачунарство настоји да угрози одређене класичне криптографске алгоритме [16].

Иако су примене квантног рачунарства бројне, класично рачунарство задржава супериорност у свим осталим случајевима [13]. Ова чињеница нам указује да ће у будућности обе врсте рачунарстава да буду у употреби и да појава квантног рачунарства не представља крај за класично рачунарство. У раду [25], аутори објашњавају да квантно рачунарство није погодно за решавање проблема који укључују велике скупове података. Наиме, због потребе да се подаци учитају и претворе у квантни формат, убрзање које доноси квантно рачунарство би било изгубљено у процесу конверзије, тако да за сада класични рачунари показују боље перформансе у овом контексту.

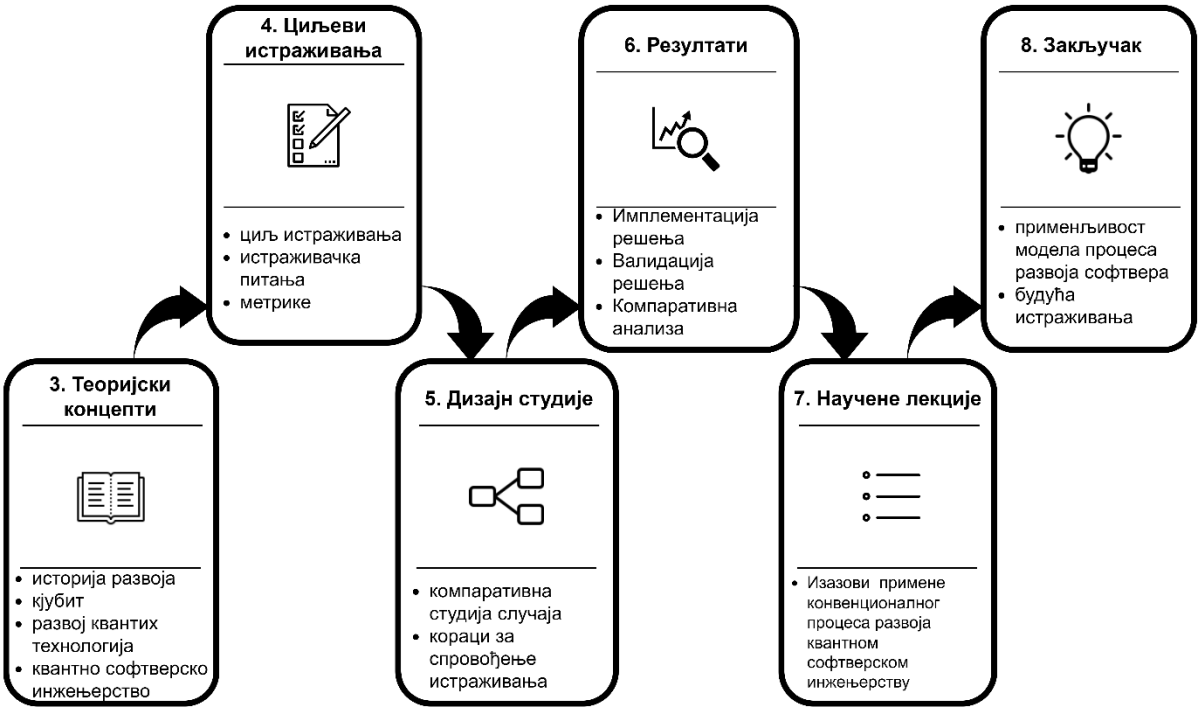
Овакво стање ствари указује на потребу за истраживањем на који начин би знање прикупљено током досадашњег рада у класичном рачунарству, тј. софтверском инжењерству, могло да се искористи у контексту квантног рачунарства. У овом раду, акценат је стављен на аспект коришћења квантног рачунарства у контексту развоја софтверских производа, које једним именом зовемо *квантно софтверско инжењерство* [18]. Овој теми је у досадашњим истраживањима посвећено недовољно пажње. Овај рад настоји да покаже у којој мери су приступи и добре праксе развоја софтвера применљива у контексту квантног софтверског инжењерства и који су то изазови који са којима се

сусрећу инжењери који имају искуство у раду у контексту класичног софтверског инжењерства када је у питању квантно софтверско инжењерство.

За потребе овог истраживања одабран је један релевантан проблем—проблем претраге неструктурираних података — чије решење је имплементирано коришћењем класичног рачунарства, а потом и применом квантног рачунарства. Компаративном анализом оба решења извучени су закључци и запажања у форми научених лекција.

2. Структура рада

Садржај који сачињава остатак овог рада подељен је у 6 секција које су визуелно приказане на слици **Слика *1***.



Слика 1. – *Структура рада*

У трећем поглављу, приказан је развој квантног рачунарства кроз историју. Поред тога, представљен је најважнији концепт квантног рачунарства – кјубит. Већи део поглавља је посвећен овом појму, где су у оквиру додатних секција приказане различите особине кјубита. Поред појма кјубита, дат је приказ појмова квантног програмирања и

квантног софтверског инжењерства. У остатку поглавља, приказано је како се квантно рачунарство развијало у различитим лидерским софтверским компанијама. На крају поглавља, описан је програмски језик *Q#*.

Поглавље 4 приказује постављени циљ рада и дефинисана истраживача питања. Поред тога, у овом поглављу су приказане метрике на основу којих би требало да се добију одговори на истраживачка питања.

У поглављу „Дизајн студије“ представљена је методологија која је коришћена у спроведеном истраживању. Поред представљене методологије, дат је детаљан приказ корака који треба да се изврше у оквиру студије.

У шестом поглављу, детаљно је описана спроведена студија. Дат је приказ имплементације решења помоћу класичног и квантног рачунарства. Након тога, приказана је валидација оба решења. На крају поглавља, објашњена је примена компаративне анализе.

Поглавље „Научене лекције“ даје структурирани приказ искуства које је прикупљено спровођењем истраживања. Научене лекције представљају изазове који су уочени приликом примене процеса развоја софтвера из класичног софтверског инжењерства у квантном софтверском инжењерству.

У последњем поглављу, приказани су закључци изведени на основу резултата спроведене студије. Поред тога, изнет је низ предлога у ком смеру би требало да се усмере будућа истраживања на ову тему.

3. Теоријски концепти

У даљем тексту, рачунаре који нису квантни рачунари називаћемо класични рачунари, а рачунарство које се не извршава на основу принципа квантне физике једним именом зовемо класично рачунарство. Како би се избегла двосмисленост и на јаснији начин приказало шта су заправо квантни рачунари, искористићемо примере из историје развоја рачунара. Пре рачунара који раде са дигиталним вредностима, коришћени су рачунари који су радили са аналогним сигналима [1]. Аналогни сигнали су континуалне физичке величине и коришћење аналогних сигнала у широком спектру задатака представља бољи избор од коришћења дигиталних сигнала. Приликом обраде аналогних сигнала јављају се проблеми као што су шум и деградација сигнала приликом обраде. Са друге стране, дигитална технологија се заснива на коришћењу дискретних вредности. Информације су репрезентоване као бинарни кодови и оваква репрезентација је отпорна на шум у великој мери и даје веома прецизне резултате. Најновија технологија, квантна технологија, представља одређену врсту споја претходне две технологије. Наиме, квантна технологија да би представила информације користи стања описана квантном физиком. Међутим, пре него што је измерено, стање кјубита карактеришу континуалне вредности. У поређењу са аналогном технологијом, где је читање вредности могуће у сваком тренутку, читање вредности у квантном рачунарству узрокује губитак континуалне вредност стања и доводи до његовог свођења на нулу или јединицу. Особина квантних битова да приликом читања губе континуалну компоненту представља изазов у раду са овом технологијом.

3.1. Историја квантног рачунарства

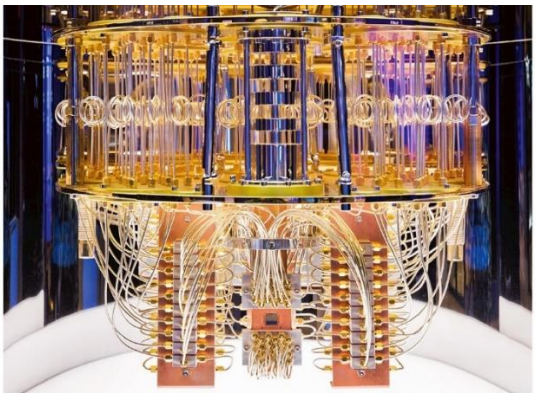
Развој квантне физике везује се за почетак 20. века кад су научници попут Макса Планка, Нилса Бора и Алберта Ајнштајна поставили основе ове гране физике [2]. Захваљујући овом достигнућу из области физике, нешто више од пола века почела је да се рађа идеја о квантном рачунарству. Идеју о рачунарском уређају заснованом на квантној механици истраживали су још 1970-их година физичари и компјутерски научници. Још 1969. Стивен Визнер је предложио квантну обраду информација као могући начин за боље постизање криптографских задатака. Ипак, прва четири објављена рада о квантним информацијама припадају Александру Холеву (1973), Р. П. Поплавском (1975), Роману Ингардену (1976) и Јурију Манину (1980) [4].

Године 1982, физичар Ричард Фејнман је предложио да квантномеханички феномени могу да се користе за симулацију квантног система ефикасније од наивне симулације на класичном рачунару. Током 1993., Бернстаин и Вазирани су показали да квантни рачунари могу да крше проширену Черч-Тјурингову тезу — темељни принцип информатике које су говориле да су перформансе свих рачунара само полиномијално бржи од пробабилистичке Тјурингове машине [27]. Њихов квантни алгоритам је понудио експоненцијално убрзање у односу на било који класични алгоритам за одређени рачунарски задатак [14]. Још један пример који показује експоненцијално убрзање квантног алгоритма је дело Дана Симона [4]. Он је показао да је квантно рачунарство једини модел рачунарства који нарушава проширену Черч-Тјурингову тезу и стога само квантни рачунари су способни за експоненцијална убрзања у односу на класичне компјутери [14]. Године 1994. Петер Шор [28] је показао да неколико важних рачунских проблема могу знатно ефикасније да се реше коришћењем квантног рачунара — када би таква машина могла да се конструише. Конкретно, он креирао алгоритам за решење проблема факторизације великих целих бројева и брзо решавање дискретних логаритама [28]. Ови проблеми не би могли да буду решени коришћењем најјачих рачунара данашњице током хиљада или милиона година. Ово је било веома значајно откриће, будући да је сугерисало да свако ко има квантни рачунар може да разбије данашње стандардне криптографске заштите, као што је *SHA256* алгоритма. Ови резултати су подстакли истраживаче за развој других квантних алгоритама који имају експоненцијално боље перформансама од класичних алгоритама, као и да покушавају да

створе основне градивне блокове од којих би квантни компјутер могао да се направи. [16]

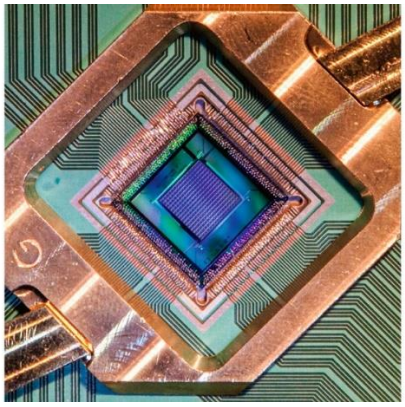
3.2. Основни концепти квантног рачунарства

Под појмом квантног рачунарства дефинише се рачунарство које се извршава на квантним рачунарима. Аутори у раду [14] квантне рачунаре дефинишу као уређаје који користе специфична својства квантне механике за обављање прорачуна.

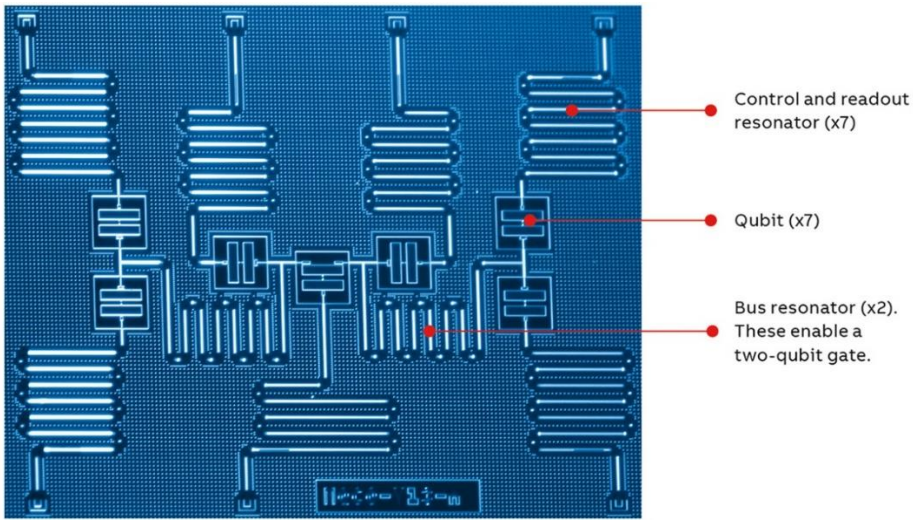


Слика 2. - *Квантни рачунар (слика преузета са [24])*

У основи квантних рачунара се налази квантни процесор – компонента која треба да врши обраду информација. Основна јединица информације у квантном рачунарству се назива кјубит. Обрада информација у квантном процесору је остварена манипулацијом над кјубитима [15]. У овом поглављу дат је преглед кјубита и важних особина кјубита који омогућавају решавање проблема који нису били могући помоћу класичних рачунара.



Слика 3. – *Квантни чип D-Wave 2000Q компаније ИБМ који има 2048 кјубита (слика преузета са [24])*



Слика 4. – *Квантни процесор компаније ИБМ са 7 кјубита (слика преузета са [24])*

Појмови дефинисани у даљем тексту овог одељка користе рад [14] као основну референцу, и уколико није другачије назначено, подразумева се да текст користи наведени рад као референцу.

3.2.1. Кјубит

Кјубит представља основну јединицу информације у квантном рачунарству. За разумевање кјубита важно је разумевање појма бита – основне јединице информације у класичном рачунарству. У једном биту може да се сачува информација у виду једне од две вредности – 0 или 1. Кјубит, као и бит, када прочитамо његову вредност, може да се нађе у једном од два стања – 0 или 1. Међутим, пре него што прочитамо вредност кјубита, он може да се нађе и у стању на два нивоа, односно у линеарној комбинацији ова два бинарна стања. Овај појам да кјубит може да се нађе у комбинацији два стања називамо суперпозиција. Чињеница да тада можемо да изводимо прорачуне над подацима у суперпозицији та два стања је један од примарних извора потенцијалних предности квантног рачунарства у односу на класично рачунарство [15].

За приказ стања кјубита користимо дводимензионални вектор, код ког збир квадрата његових вредности мора да буде једнак 1. Овај вектор, назван вектор квантног стања, садржи све информације потребне за описивање квантног система са једним кјубитом [17]. Сачињен је од два јединична вектора која, чијом линеарном комбинацијом могу да се добију сви остали вектори који репрезентују стање квантног система. У примеру који следи је приказан вектор који репрезентује кјубит:

$\begin{bmatrix}\alpha\\ \beta\end{bmatrix}$, ако важи да су α и β комплексни бројеви код којих је

$$|\alpha|^2 + |\beta|^2 = 1.$$

Примери валидних квантних стања би били:

$$\begin{bmatrix}0\\ 1\end{bmatrix}, \begin{bmatrix}1\\ 0\end{bmatrix}, \begin{bmatrix}\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}}\end{bmatrix} \text{ и слично.}$$

Вектор $\begin{bmatrix}x\\ y\end{bmatrix}$ можемо да напишемо и као:

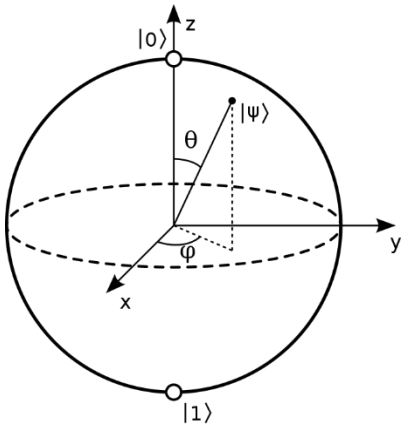
$$x\begin{bmatrix}1\\ 0\end{bmatrix} + y\begin{bmatrix}0\\ 1\end{bmatrix}.$$

У литератури се често као алтернатива векторској нотацији користи бра-кет (енг. *bra-ket*) нотација Пола Дирака [29]. У овој нотацији, јединични вектори се приказују као:

$$\begin{bmatrix}1\\ 0\end{bmatrix} = 0\rangle \text{ и } \begin{bmatrix}0\\ 1\end{bmatrix} = 1\rangle.$$

Коефицијенти *x* и y се називају амплитуде. Коефицијент x означава вероватноћу да ће се након мерења стања кјубита добити вредност 0, док коефицијент *y* означава вероватноћу да ће измерена вредност да буде једнака 1.

Како би појам кјубита био јасније објашњен, често се за његову репрезентацију користи нотација која се назива Блох сфера (енг. *Bloch Sphere*) [14]. Захваљујући Блох сфери, кјубите је могуће приказати у тродимензионалном простору. Важно је поменути да се ова нотација користи искључиво за описивање квантног стања са једним кјубитом. За полове Блох сфере изабрани су јединични вектори 0⟩ и 1⟩. Стрелице на слици показују правац у коме је вектор квантног стања усмерен и свака трансформација стрелице се може посматрати као ротација око једне од координатних оса. На слици **Слика 5**.је приказана нотација помоћу Блох сфере.



Слика 5. - *Приказ кјубита помоћу Блох сфере (слика преузета из [14])*

Било које стање у квантном рачунарству може да се представи као вектор који почиње у координатном почетку и завршава се на површини јединичне Блох сфере. Као конвенција, узима се да су полови сфере јединични вектори 0⟩ и |1⟩.

3.2.2. Суперпозиција

Сагласно постулатима квантне механике, системи приказују одређено стање тек након што су измерени. Пре вршења мерења, системи се налазе у неодређеном стању; након мерења, системи прелазе у конкретно стање. Ако разматрамо систем који, на пример, може заузети једно од два дискретна стања по обављеном мерењу, ова стања су означена Дираковом нотацијом као |0⟩ и |1⟩. Ова стања представљају еквивалент класичним стањима 0 и 1. При томе, могуће је приказати суперпозицију стања као линеарну комбинацију наведених стања, изразом:

$$u|0\rangle + v|1\rangle,$$

где су u и v параметри који означавају релативни удео сваког стања у суперпозицији. Другим речима, u и v означавају вероватноћу да ће кјубит након мерења да има вредност измерену као 0⟩, односно |1⟩.

3.2.3. Операције над једним кјубитом

Како би обрада информација у квантом рачунарству била могућа, неопходно је вршити промену стања квантних битова. У овом одељку је дат преглед неколико унарних оператора (енг. *gates*), односно оператора који се примењују над једним кјубитом. Операторе другим именом називамо логичка кола. Важно је напоменути да су сви унарни оператори осим мерења реверзибилни, односно уколико се примене двапут, враћају кјубит у првобитно стање пре него што су примењени.

Оператори се најчешће приказују коришћењем одговарајућих симбола који их репрезентују. Друга, веома честа и важна нотација јесте нотација помоћу матрица. [29] Наиме, будући да стања су кјубита приказана као вектори, оператори који трансформишу стања су репрезентовани помоћу матрица. У наставку је дат преглед често коришћених оператора.

- ***NOT* оператор –** овај оператор представља најједноставнији оператор. Он представља аналогни оператор оператору *NOT* у класичном рачунарству. Ако је

кјубит у стању |0⟩, овај оператор га преводи у стање |1⟩ и обрнуто. На слици **Слика *6*** је приказан оператор *NOT*.

Слика 6. - *NOT оператор*

Матрица која репрезентује *NOT* оператор је приказана у наставку.

$$\mathrm{NOT} \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Примена оператора се извршава тако што тако што се изврши множење матрице вектором:

$$\mathrm{NOT}\begin{pmatrix} p \\ q \end{pmatrix} \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{pmatrix} p \\ q \end{pmatrix}.$$

- ***Z* оператор** – користи се за мењање фазног стања кјубита. У класичном рачунарству не постоји аналогни оператор. Када је примењен, уколико је кјубит у стању |0⟩, оператор *Z* га не мења. Са друге стране, уколико је стање кјубита |1⟩, оператор *Z* мења фазу тог стања за 180 степени (постаје -1). На слици **Слика *7*** је приказана нотација оператора *Z*.

Слика 7. - *Z оператор*

Матрична репрезентација овог оператора је приказана у наставку.

$$Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Поред *Z* оператора, постоје аналогни *X* и *Y* оператори који мењају фазу кјубита у односу на одговарајућу осу која се налази у имену оператора.

- **Хадамард *(*енг. ***Hadamard*****) оператор –** најзначајнији оператор у квантном рачунарству. Његовом применом, кјубит који се налазио у основном стању прелази у стање униформне суперпозиције, односно стања у ком би се уколико би се извршило мерење, ја једнаком вероватноћом појавила вредност $|1\rangle$, односно $|0\rangle$. Овај оператор представља први корак у многим познатим квантним алгоритмима. На слици **Слика *8*** је представљена нотација овог оператора:

Слика 8. - *Хадамард оператор*

Матрична репрезентација овог оператора је:

$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

3.2.4. Вишекјубитни систем

Уколико би операције са једним кјубитом биле једина врста операција у квантном рачунару, тада би обичан калкулатор надмашио његову рачунску моћ. Права снага квантног рачунарства постаје очигледна тек када повећате број кјубита. Као што је описано у претходним одељцима, кјубити имају особине попут способности да буду истовремено у више од једног стања. Међутим, моћ квантног рачунарства произилази, делом, из димензије векторског простора квантних стања која експоненцијално расте са бројем кјубита. Ово значи да док појединачни кјубит може једноставно да се моделује, симулација квантног рачунања са педесет кјубита би вероватно превазишла границе постојећих суперкомпјутера. Повећање величине рачунања само за један додатни кјубит удвостручује меморијске капацитете за чување стања и отприлике удвостручује време потребно за рачунање. Овај брзи пораст рачунске моћи је разлог због ког квантни рачунар са релативно малим бројем кјубита може значајно да надмашује најмоћније суперкомпјутере данашњице, али и сутрашњице [17].

Ако посматрамо систем од два бита, у класичном рачунарству, два бита могу да имају четири могућа стања – 00, 01, 10 и 11. Да би се израчунао излаз двобитне Булове функције за сваки од ових могућих улаза, било би потребно генерисати сваки пар сигнала и затим их или редом слати у оператор који одговара функцији. С друге стране, ако би се користио квантни рачунар, све четири могућности би биле садржане у стању два кјубита путем суперпозиције четири квантна основна стања $|00\rangle$, $|01\rangle$, $|10\rangle$ и $|11\rangle$. Израчунавање би било извршено коришћењем само једног квантног оператора, који би радио над свим стањима паралелно, истовремено. Још један начин који сведочи о потенцијалној снази скупа кјубита јесте пример количина информација потребне да се потпуно одреди стање система кјубита. Конвенционални дигитални двобитни систем захтева два бита информације да представи своје стање. Насупрот томе, систем од два кјубита постоји у суперпозицији четири стања ($|00\rangle$, $|01\rangle$, $|10\rangle$ и $|11\rangle$), захтевајући четири комплексне константа (a_{00}, a_{01}, a_{10}, a_{11}) да би се потпуно описало квантно стање, уместо два бита. Различите вредности ова четири коефицијента кодирају резултате свих могућих врста претходних операција изведених над два кјубита, као и вероватноћу да се заврши у сваком стању ако се систем измери. На основу овог примера, принцип може да се генерализује и да се каже да помоћу N кјубита можемо да моделујемо систем са 2^N коефицијената, тј. аналогно посматрано заправо имамо систем са 2^N бита информације насупрот класичном рачунару где нам N бита омогућава да кодирамо N бита информације.

Још један важан појам у контексту вишекјубитних система јесте релативна фаза. Релативна фаза је концепт у квантној механици који се односи на фазни угао или фазни однос између различитих квантних стања. У квантном рачунарству, релативна фаза игра важну улогу у процесима интерференције између квантних стања и може имати значајне ефекте на резултате квантних операција. У контексту квантних алгоритама, квантна суперпозиција се користи за постизање бољих перформанси у односу на класичне алгоритме. Промена релативне фазе током квантних операција може да утиче на крајње резултате квантних алгоритама и оптимизација фаза је важан компонента дизајна ефикасних квантних алгоритама.

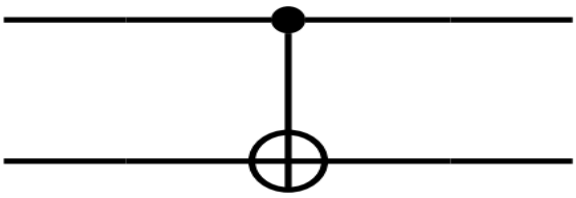
У контексту вишекјубитних система појављује се особина која се назива квантно спрезање (енг. *Entanglement*). Овај појам се дефинише као специјално стање у ком се налазе два или више кјубита и у ком мерење стања једног кјубита је у корелацији са стањем другог кјубита. Спрегнути кјубити су у корелацији на начин да се не могу

описати независно један од другог. То значи да било која операција која се дешава на стању једног кјубита у сплетеном пару такође утиче и на стање другог кјубита.

3.2.5. Операције у вишекјубитном систему

Попут операција које су карактеристичне за појединачне кјубите, важно је разумевање операција које се извршавају на неколико кјубита. У овом поглављу представљене су одабране операције важне за разумевање материјала представљеног у раду.

- ***CNOT* оператор** – представља један од основних оператора у квантном рачунарству. Овај оператор се примењује над два кјубита и функционише на начин да делује на циљни кјубит на основу стања контролног кјубита. Овај оператор се често називају квантном верзијом класичних *XOR* оператора у класичном рачунарству, будући да уводи операцију спрезања која ствара специфичан однос између контролног и циљног кјубита. Уколико је контролни кјубит у стању $|1\rangle$, мења се стање циљног кјубита. Ако је контролни кјубит у стању $|0\rangle$, циљни кјубит остаје неизмењен. Комбинација H оператора над контролним кјубитом и *CNOT* оператора над контролним и циљним кјубитом доводи кјубите у стање спрезања (енг. *Entanglement).* На слици **Слика 9** је представљена нотација *CNOT* оператора.

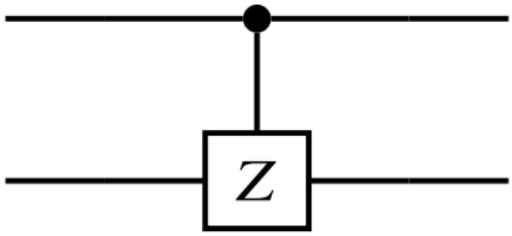


Слика 9. - *CNOT оператор*

Матрица која репрезентује *CNOT* оператор је дата у наставку.

$$CNOT \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

- ***CZ* оператор** – један пример контролисаног оператора, код ког се на основу стања контролног кјубита примењује оператор *Z* над циљним кјубитом. На слици **Слика *10*** је приказан *Z* оператор.

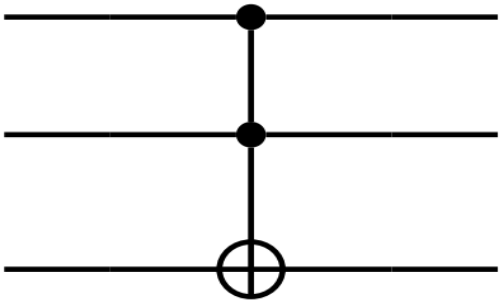


Слика 10. - *CZ оператор*

Матрична репрезентација је дата у наставку.

$$CNOT \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

- ***CCNOT* оператор** – такође познат као Тофоли оператор (енг. *Toffoli gate*), представља оператор који оперише над три кјубита. Функционише као контролисано-контролисани *NOT* оператор. Он представља проширење *CNOT* оператора и уводи додатни контролни кјубит, што га чини кључном компонентом у квантним колима за имплементацију различитих алгоритама и техника корекције грешака. *CCNOT* оператор изводи *NOT* операцију на циљном кјубиту ако и само ако су оба контролна кјубита у стању $|1\rangle$. Другим речима, он мења стање циљног кјубита само када су оба контролна кјубита у $|1\rangle$ стању; у супротном, оставља циљни кјубит непромењеним. На слици **Слика *11*** је представљена нотација *CCNOT* оператора.



Слика 11. - *CCNOT оператор*

Матрица која репрезентује *CCNOT* оператор је дата у наставку.

$$CNOT \equiv \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

3.3. Квантно програмирање

Квантна логичка кола описана у претходним одељцима представљају основне компоненте помоћу којих је могуће изградити квантне алгоритме. Квантни алгоритми, као и алгоритми у класичном рачунарству, представљају секвенцу операција које решавају одређени проблем или групу проблема. Ако квантне алгоритме посматрамо са становишта физичке репрезентације, можемо да уочимо битну разлику у односну на класичне алгоритме. Наиме, будући да су класични алгоритми сачињени од дискретних компоненти, свака од тих компоненти засебно извршава одређену операцију примајући улазне податке и креирајући излазне податке. Са друге стране, кјубите у квантним алгоритмима можемо да посматрамо као еквивалент компонентама у аналогним, односно механичким системима. Они су физички повезани тако да можемо да их посматрамо као јединствену целину која је тако дизајнирана да решава одређени проблем.

До сада је позната неколицина квантних алгоритама који демонстрирају супериорност квантних технологија у решавању одређених проблема у односу на класично рачунарство. Два најпознатија алгоритма су Гроверов (енг. *Grover's algorithm*) и Шоров (енг. *Shor's*) алгоритам. Шоров алгоритам је алгоритам који за дати цео број *N* може да пронађе његове просте чиниоце и при томе постиже далеко боље перформансе од свих познатих класичних алгоритама. Када је у питању Гроверов алгоритам, будући да је овај алгоритам коришћен за потребе овог рада, у наставку је детаљно описан Гроверов алгоритам.

3.3.1. Гроверов алгоритам

Гроверов алгоритам је квантни алгоритам који решава проблем претраживања несортиране базе података у полиномијалном времену $O(\sqrt{N})$ користећи $O(\log N)$ меморијског простора. Формулисан је од стране научника Лова Гровера (енг. *Lov Grover*) 1996. године. У класичном рачунарству, проблем претраге неструктуриране базе података је захтевао примену принципа исцрпне претраге (енг. *Brute force*) помоћу ког би се пролазило редом кроз елементе како би се утврдило да ли се тражени елемент налази у колекцији. Овакви алгоритми имају линеарну сложеност извршавања, односно $O(N)$. Гроверов алгоритам доноси експоненцијално убрзање и превазилази моћ класичних алгоритама [14].

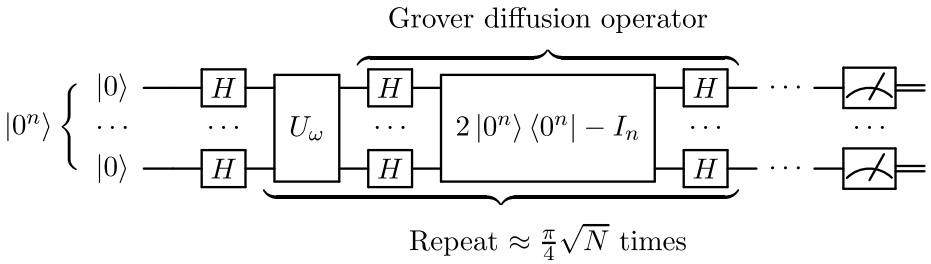
Како бисмо објаснили на који начин Гроверов алгоритам постиже изузетне перформансе, прво треба установити како репрезентујемо колекцију података, односно базу података која треба да се претражи. Као колекцију података узимамо скуп свих могућих стања у којима кјубити могу да се нађу, односно у случају три кјубита, то би била стања $|000\rangle$, $|001\rangle$, …, $|111\rangle$. Вредност коју тражимо је једна вредност, односно једно стање, у скупу свих стања. Главни принцип који се користи у Гроверовом алгоритму је амплификација амплитуде (енг. *Amplitude amplification*). Овај принцип се заснива на томе да настоји да повећа вероватноћу читања траженог стања, при чему уједно смањује вероватноће читања осталих стања.

Након разумевања основних главних принципа коришћених у алгоритму, у наставку је представљен низ корака који сачињавају алгоритам:

1. Први корак у имплементацији Гроверовог алгоритма представља превођење у суперпозицију свих кјубита који се налазе у регистру. Превођење у суперпозицију се постиже тако што се на сваки кјубит примењује Хадамардов оператор. Овим кораком се постиже стање *униформне суперпозиције*.
2. Други корак представља креирање црне кутије (енг. *Oracle*), односно функције код које не знамо детаље имплементације, већ само улазне и излазне параметре. Црна кутија коју креирамо треба да означи стање које желимо да пронађемо. Ова операција функционише тако што извршава фазни обрт амплитуде траженог стања. Црна кутија се имплеметира на различите начине, зависно од тога које стање проверавамо да ли постоји и колико стања репрезентује базу података.

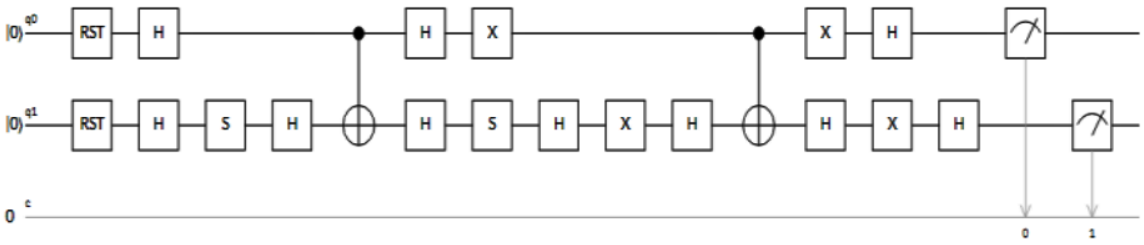
3. Наредни корак представља креирање таквог кола које ће да повећа амплитуду стања које тражимо, а уједно да смањује амплитуде осталих стања. Овај оператор се назива Гроверов дифузиони оператор (енг. *Grover's diffusion operator*). Поменуто понашање оператор постиже тако што ради инверзију амплитуде у односу на средњу вредност – уколико је нека амплитуда имала вредност *x* изнад средње вредност, након примене дифузионог оператора имаће вредност *x* испод средње вредности. Овај корак представља аплификацију аплитуде.
4. Кораци 2. и 3. се понављају $\sqrt{N}$ пута, при чему *N* представља број елемената у бази.
5. Као последњи корак, врши се мерење квантног система. Резултати мерења су вероватноће да ће систем да се нађе у одређеном стању. Након извршених итерација у претходним корацима, вероватноћа траженог стања треба да буде далеко већа од осталих вероватноћа.

Приказ корака Гроверовог алгоритма дат је на слици:



Слика 12. – *Гроверов алгоритам*

У раду [31] дат је приказ Гроверовог алгоритма у двокјубитном систему, где се проверава постојање стања |10⟩. Дизајн алгоритма је приказан на слици **Слика *13***.



Слика 13. – *Гроверов алгоритам за двокјубитни систем где се проверава постојање стања |10⟩ (слика преузета из [31])*

Оператор *S* који се појављује у представљеном примеру означава *Z* оператор који врши промену фазе за 90 степени. У првом кораку алгоритма се стања доводе у униформну суперпозицију применом два Хадамард оператора. У том тренутку, регистар се налази у следећем стању:

$$|\psi\rangle \rightarrow \frac{1}{2}(|00\rangle + |01\rangle - |10\rangle + |11\rangle) \ .$$

Наредни корак подразумева извршавање црне кутије, која инвертује вредност амплитуде траженог стања. Након примере два Хадамард оператора, добије се следеће стање:

$$|\psi\rangle \rightarrow \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle + |11\rangle) \ .$$

Применом фазног помераја, ново стање регистра је:

$$|\psi\rangle \rightarrow \frac{1}{2}(|00\rangle - |01\rangle - |10\rangle - |11\rangle) \ .$$

На крају извршавања, кад се примене оператори мерења, резултати показују 100% вероватноћу да систем након мерења заузме стање |10⟩.

3.4. Квантно софтверско инжењерство

Развојем квантног рачунарства, како алгоритама тако и хардверске подршке, природним током настаје потреба да се технологија искористи у циљу решавања

проблема. Начин на који се технологија примењује јесте кроз креирање софтверског производа коришћењем одређене технологије. Како се рачунари користе у решавању бројних проблема у свакодневном животу више од неколико деценија, успостављен је процес развоја софтверских производа који једним именом називамо софтверско инжењерство. У овом поглављу, дискутовано је о томе који је резултат примене софтверског инжењерства у контексту квантног рачунарства и које изазове таква примена доноси. У даљем тексту, за софтверско инжењерство у контексту класичног рачунарства користићемо термин „софтверско инжењерство“, док у контексту квантног рачунарства користимо термин „квантно софтверско инжењерство“.

Софтверско инжењерство представља дисциплину која за крајњи циљ има креирање софтверског производа који решава проблем из реалног света за који је креиран [34]. Софтверски производ креиран је коришћењем једног или више програмских језика, и тако креиран артефакт се извршава на одређеном хардверу и корисницима омогућава коришћење. Процес, током ког се проблем из реалног света трансформише у идеју, на основу које настаје производ једним именом називамо процес развоја софтвера. Током историје, овај процес се мењао, усавршавао и даље наставља да се усавршава у циљу креирања бољих производа и оптималног искоришћења ресурса. Постоји више варијација процеса развоја софтвера који су прихваћени од стране заједнице као најбољи[35]. У овом раду, посматраћемо процес развоја који садржи следеће фазе:

- анализа захтева корисника / спецификација захтева решења,
- дизајн решења,
- имплементација решења,
- дебаговање решења,
- тестирање / валидација решења и
- постављање решења на продукциони хардвер.

Прва фаза, односно *фаза спецификације решења* [26] је представља низ корака у оквиру који је потребно трансформисати захтеве корисника у низ функционалности које крајњи производ треба да омогући. Овај део процеса захтева добро разумевање могућности које технологија омогућава од стране особе која ради на овом задатку.

У оквиру друге фазе, односно *фазе дизајна решења*, на основу низа функционалности које треба да буду испуњене, потребно је одабрати технологије и алате помоћу којих ће софтверски производ да буде креиран. Будући да је током историје софтверског инжењерства развијен велики број програмских језика, радних оквира који

олакшавају рад са програмским језицима, као и велики број додатних алата који су део процеса развоја софтвера, потребно је да се донесе одлука и изврши одабир најбољих алата. Како би особа могла да донесе праву одлуку, потребно је одлично познавање доступних могућности и њихових карактеристика како би софтверски производ био креиран на најбољи могући начин и имао најбоље могуће перформансе.

У оквиру *фазе имплементације* потребно је направити производ који испуњава све захтеве одређене у првој фази коришћењем технологија специфицираних у другој фази. Особа која ради на овом делу треба добро да познаје одабране технологије и да њиховим коришћењем направи тражени производ. Када је производ спреман, требало би проверити да ли постоје одређени дефекти.

Први корак провере дефеката јесте у оквиру *фазе дебаговања*, где особе које су правиле производ уз помоћ софтверских алата отклањају примећене грешке. Други део валидације решења је *фаза тестирања*, у оквиру које би требало да се производ користи на начин на који се очекује да ће корисници да га користе. Уколико одређене грешке буду уочене, производ се враћа у претходну фазу где би требало дебаговањем да се грешке отклоне. Када је утврђено да производ ради како треба и нема дефеката, решење је потребно поставити на хардвер тако да се крајњим корисницима омогући његово коришћење. Због доступности великог броја хардвера, од сервера на захтев (енг. *on-premise*) до клауд решења, потребно је познавање свих опција како би се одабрало најбоље решење.

По узору на претходно дискутовано софтверско инжењерство, настале су различите иницијативе за примену софтверског инжењерства у контексту квантног рачунарства. У раду [18] аутори констатују да је за успех квантног рачунарства потребан настанак квантних софтверских производа, односно решења која би била примењена у реалном свету. Они предвиђају да би у неколико наредних година требало да се догоди настанак таквих производа. У оквиру овог рада помиње се „Талавера манифест“ (енг. *Talavera Manifesto*). Он представља 9 принципа на којима би требало да се заснива квантно софтверско инжењерство и овај манифест је настао као резултат дискусије више од 100 представника академске заједнице и људи из индустрије. Принципи у манифесту су:

1. независност од квантних програмских језика и технологија,
2. промовише коегзистенцију класичног и квантног рачунарства,
3. подржава управљање пројектима развоја квантног софтвера,
4. разматра еволуцију квантног софтвера,

5. настоји да обезбеди квантне програме без дефеката,
6. обезбеђивање квантног софтвера,
7. промовише поновну употребу квантног софтвера,
8. наглашава безбедност и приватност по дизајну,
9. покрива управљање софтвером.

Аутори у раду [19] констатују да квантно софтверско инжењерство још увек неразвијено и истичу неколико праваца у којима би могло да се развија. Први приступ истиче да би квантно рачунарство требало да се прилагоди постојећим парадигмама и класичног софтверског инжењерства. Други правац подстиче развој новог приступа који би се разликовао од конвенционалног процеса развоја.

О разликама између квантног и класичног рачунарства, као и квантног и класичног софтверског инжењерства дискутују аутори у раду [20]. У табели Табела 1 је дат приказ разлика које аутори наводе у раду:

Табела 1. – *Разлике између класичног и квантног рачунарства (табела преузета из [20])*

Аспект	**Класично рачунарство**	**Квантно рачунарство**
Кодирање информација	Бит	Кјубит
Градивни блокови	Бинарне операције над бинарним улазима	Квантни оператори
Логичке функције	Над регистром од *n* битова	Над регистром од *n* кјубита
Стање	0 или 1 за један бит. На пример, два бита задржавају тачно једну вредност у исто време из следећих могућих вредности 00, 01, 10, 11.	Суперпозиција: квантни програм може да се нађе у више стања истовремено; Квантно спрезање: два кјубита или регистра деле исто стање. На пример, два кјубита могу задржати четири вредности одједном: 00, 01, 10 и 11.
Процес програмирања	Мења се при употреби различитих програмских језика, простирући се од језика на ниском нивоу (асемблера) до језика на вишем нивоу, на пример, Пајтона	Тренутна пракса се одвија на језику асемблера или дизајнирањем квантних кола
Тестирање и дебаговање	(1) читање међувредности је могуће у одређеним	(1) читање међустања уништава суперпозицију; (2) већина

	случајевима; (2) неки механизми за утврђивање да ли је тест прошао су пробабилистички; (3) мало хардверских грешака; (4) могуће је постављати тачке прекида за интерактивно дебаговање	механизама за утврђивање да ли је тест прошао су пробабилистички; (3) честе грешке у хардверу; (4) није могуће је постављати тачке прекида за интерактивно дебаговање (5) Суочавање са проблемом декохеренције
Развој софтвера	(1) Многи модели процеса развоја, као што су агилни приступ и модел водопада; (2) пуно доступних радних оквира за прелазак са дизајна на виском нивоу на ниже нивое; (3) интуитивни програмски језици	(1) нема установљеног процеса развоја; (2) мањак апстракције – не постоје механизми за превођење дизајна на вишем нивоу на ниво логичких кола (3) мање интуитивни програмски језици

У раду [20], аутори дају преглед разлика између одређених фаза развоја софтверског производа. На пример, у фази спецификације захтева, они истичу да је ова фаза прилично слична са приступом који се примењује у класичном развоју. Наглашавају да поред сличности, квантно рачунарство доноси нове изазове. Када је реч о фази тестирања, аутори наглашавају да је због чињенице да се приликом читања вредности губи податак о стању кјубита, тестирање у квантном рачунарству представља посебан изазов. Када је реч о квантном дебаговању, иако се јавља сличан проблем као код фазе тестирања, направљени су одређени алати који омогућавају праћење вредности техникама као што су трагање уназад, а такође постоје и визуализатори који омогућавају приказ стања разумљив човеку.

Иако постоје радови који се баве разликом између класичног и квантног софтверског инжењерства, свеобухватна анализа квантног процеса развоја софтвера остаје недовољно истражена тема.

3.5. Развој квантних технологија

3.5.1. Водеће компаније и квантне технологије

Током протеклих неколико деценија, истраживање у контексту квантних технологија је напредовало до тачке где су изграђени веома једноставни квантни рачунари који су временом напредовали и постајали све сложенији. О потенцијалу квантних рачунара сведочи чињеница да су лидери у свету информационих технологија попут ИБМ-а, Гугла и Мајкрософта започели истраживања у циљу креирања комплексних квантних рачунара. У даљем тексту су дати примери развоја квантних технологија које су оствари водеће ИТ компаније.

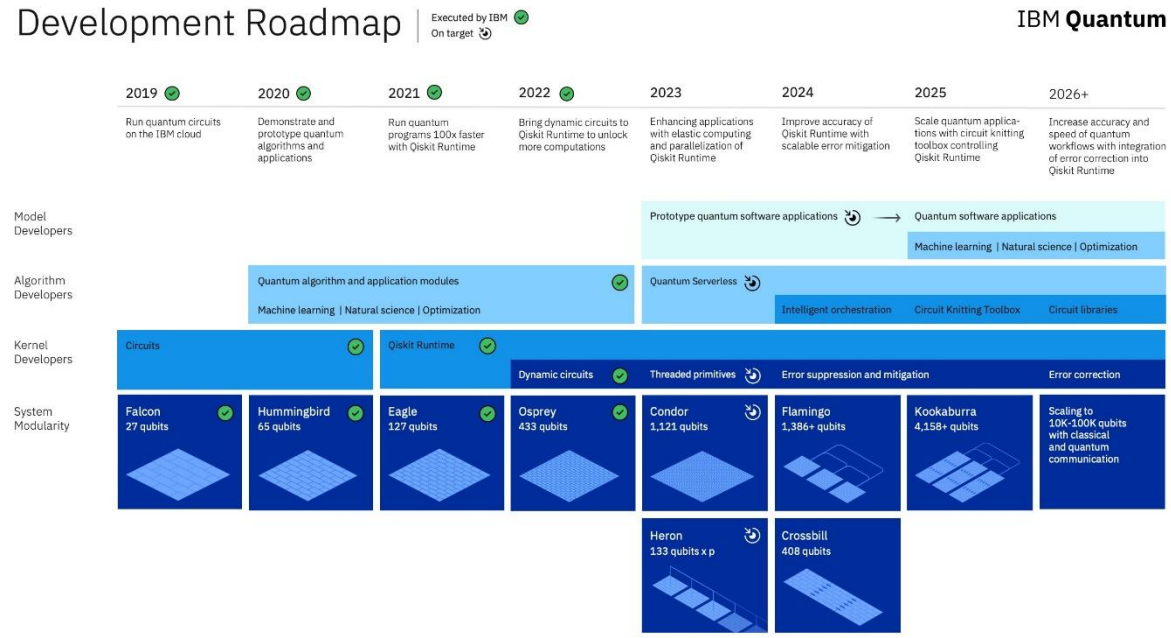
У табели **Табела *2*** је приказан развој квантних рачунара компаније Гугл. На основу приказаних података, видимо да се у последњој деценији значајно повећао број кјубита у квантним рачунарима које је представила компанија Гугл. Такође, чињеница која сведочи о томе да ће квантни рачунари моћи да буду искоришћени у реалном свету јесте да су квантни рачунари коришћени за различите хемијске симулације.

Табела 2. – *Развој квантних технологија у компанији Гугл*

Година	**2009**	**2016**	**2018**	**2019**	**2020**
Достигнуће	Демонстрација система засновано на квантним технологијама за препознавање и сортирање облика на сликама и видео записима [32]	Симулација молекула водоника коришћењем 9 кјубита [5]	Тестирање квантног рачунара са 72 квантна бита [6]	Тврдња о квантном рачунару који за 20 секунди решава проблем за који би требало 10.000 година класичном рачунару [7]	Највећа хемијска симулација на квантном рачунару

Када је реч о компанији ИБМ, она представља апсолутног лидера у свету квантне технологије. На слици **Слика *14***.је приказан развојни пут квантних технологија у оквиру компаније ИБМ. На основу приказаног можемо да закључимо да се сваке године број кјубита у квантним рачунарима удвостручује и да је тенденција да се након 2026. направе рачунари са више од 10,000 кјубита. Помоћу таквих рачунара, најсложенији квантни

алгоритми би могли да буду извршени и промене начин функционисања информационих технологија какав данас познајемо.



Слика 14. - *историја развоја квантних рачунара у компанији ИБМ [9]*

Компанија ИБМ је 2016. године на своју клауд платформу поставила први квантни рачунар који могао да изврши операције над 5 кјубита. [21] Захваљујући овој платформи, свим корисницима је омогућено да код извршавају на правим квантним рачунарима. Квантни програми који се извршавају на ИБМ-овом клауду се креирају коришћењем радног оквира Кисит (енг. *Qiskit*). Овај радни оквир нуди могућност писања кода помоћу програмског језика Пајтон и укључује низ библиотека које поједностављују процес писања кода.

Компанија Мајкрософт је, попут компаније ИБМ, развила могућност коришћења квантних рачунара коришћењем платформе Ежр (енг. *Azure*). Мајкрософт је, по угледу на популаран програмски језик *C#* развио квантни програмски језик *Q#*. Овај језик је тако дизајниран да корисницима омогући једноставније програмирање квантних рачунара уз акценат на повећању апстракције и сакривања хардверских детаља од крајњег корисника. У оквиру пакета за развој софтвера који Мајкрософт нуди, поред програмског језика *Q#,* укључен је компајлер и симулатор за извршавање програма писаних овим језиком. Захваљујући симулатору, корисници су у могућности да програме написане помоћу *Q#* извршавају на сопственим рачунарима.

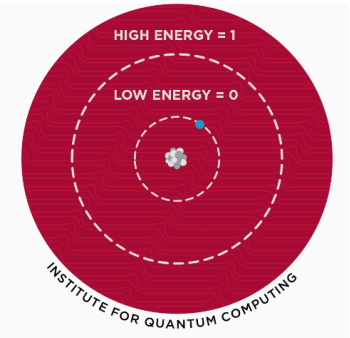
3.5.2. Технологија квантних процесора

У претходним одељцима било је речи о кјубитима и њиховим карактеристикама. У овом поглављу, акценат је на објашњењу физичке структуре кјубита. Биће представљено који се то материјали из природе користе да се направи један физички кјубит и биће објашњене карактеристике тих материјала. Примери технологије на којима су засновани квантни рачунари, односно квантни процесори су кјубити засновани на суперпроводним колима као и кјубити засновани на заробљеним јонима. [3]

Када су у питању кјубити засновани на суперпроводним колима. Принцип по ком функционишу је да када се охладе на ниску температуру, неки материјали омогућавају проток електричне струје без отпора. Такве материјале зовемо суперпроводницима. Могуће је дизајнирати електрична кола на бази суперпроводника тако да се понашају као кјубити. За разлику од других примера кјубита, ови системи су направљени од милијарде атома, али ипак се понашају као један квантни систем. Један начин на који можемо изградити суперпроводни кјубит је да доделимо вредност смеру у коме струја тече око електричног кола. [11]

Слика 15. - *приказ кјубита заснованом на суперпроводном колу (слика преузета из [11])*

Други пример конструисања кјубита јесте тај да је могуће користити нивое енергије електрона у неутралним атомима или јонима као кјубите. У њиховом природном стању, ови електрони заузимају најниже могуће нивое енергије. Користећи ласере, можемо да их побудимо на виши ниво енергије. Можемо доделити вредности кјубитима на основу њиховог стања енергије. [12]



Слика 16. - *приказ кјубита који користи нивое електрона (слика преузета из [11])*

Када посматрамо кјубите у контексту физичке репрезентације, важно је осврнути се на појам шума који се јавља приликом рада са физичким кјубитима. Наиме, пошто се кјубит пре мерења налази у суперпозицији, његово стање може да узме једну вредност из континуалног скупа вредности. Ове вредности се у току обраде и читања мењају у одређеној мери и такву промену називамо шумом. Шум доводи до нежељених ефеката као што је погрешан излаз након обраде. Како би се спречио шум, развијени су корекциони алгоритми (енг. *quantum error correction – QEC*). Без њих, извршавање сложених квантних алгоритама би било немогуће. Ови алгоритми уносе значајан степен сложености – захтевају увођење додатних квантних битова који би се користили како би се створили прецизнији квантни битови. Квантни битови без грешака се називају логички квантни битови. Коришћење корекционих алгоритама још увек није примењено у довољној мери, будући да је број кјубита најкритичнији ресурс. [3]

3.6. Програмски језик *Q#*

Пакет за развој квантног софтвера који је креирала компанија Мајкрософт (енг. *Quantum Development Kit / QDK*) представља скуп алата који омогућавају креирање софтвера који може да се извшрава на квантном хардверу. Компоненте које сачињавају овај пакет су:

1. *Q#* компајлер,
2. *Q#* језички сервер,
3. алати за едиторе кода,
4. стандардна библиотека која дефинише основне компоненте језика,
5. додатне библиотеке које служе у сврху истраживања и рада у областима као што су машинско учење и хемијске симулације и

6. квантни симулатор који омогућава извршавање програма на класичном хардверу.

Најважнија компонента овог пакета је програмски језик *Q#*. Представља језик високог нивоа који настоји да сакрије детаље извршавања кода на квантном хардверу тако што програмерима омогућава унос познатих језичких конструкција. *Q#* је дизајниран тако да може да се извршава на различитим платформама, стога програмери нису ограничени да употребу специфичних платформи које подржавају *Q#*. Вишеплатформска подршка је могућа захваљујући начину на који се код компајлира. Наиме, уместо директног компајлирања у хардверске инструкције, програмски код се компајлира у међурепрезентацију (енг. *Quantum Intermediate Representation - QIR*). Захваљујући међурепрезентацији, да би *Q#* могао да се извршава на некој платформи, потребно је да се делимично или у целости имплементира сет инструкција *QIR*-а за одређену платформу. [15]

Будући да *Q#* представља квантни еквивалент програмском језику *C#* креираном од стране Мајкрософта, може се приметити неколико ствари које су позајмљене из тог контекста. На пример, типови пројеката које *QDK* нуди су: (1) конзолна апликација, (2) библиотека класа и (3) *xUniti* пројекат за тестирање.

Q# је дизајниран тако да подстиче и олакшава композицију писања квантног и класичног кода. У наставку је дат приказ синтаксе *Q#* програмског језика када су у питању концепти из класичног рачунарства, а потом су дати примери синтаксе у контексту квантног програмирања.

- Помоћу кључних речи **let** и **mutable** можемо да дефинишемо непроменљиву и променљиву варијаблу, респективно.

```
1    let nepromenljiva = 0;
2    mutable promenljiva = 5;
```

Листинг 1. – *Променљиве у језику Q#*

- Условно гранање могуће је променити коришћењем **if / else if / else** конструкције.

```
1    let nepromenljiva = 5;
2    if (nepromenljiva == 4) {
3      // …
4    } else if (nepromenljiva == 3) {
5      // …
6    } else {
```

```
7       // …
8     }
```

Листинг 2. - *Условно гранање у програмском језику Q#*

- Итерације кроз колекције података могуће су уз помоћ **for** петље, која представља широко примењену синтаксу у свету програмирања. Друга могућност за итерирање кроз колекцију јесте **repeat until** конструкција, која представља еквивалент **while** петљи у програмском језику *C#*. У наредом листингу дат је приказ коришћења **for** petlje.

```
1     let nonModifiableCollection = [2,3,4];
2     for index in 0 .. Length(nonModifiableCollection) - 1 {
3       // …
4     }
```

Листинг 3. – *Пример петље у програмском језику Q#*

У претходном листингу, приказана је употреба уграђене функције **Length** која као резултат враћа број елемената у колекцији. Поред уграђених функција, као и у сваком програмском језику, и у *Q#* постоје уграђени типови података. Неки од примера типова података су **Int, Bool, Double** и слично. Постоје уграђени механизми за конверзију типова, као и могућност имплицитне конверзије (кастовања).

- Као и у *C#*, и *Q#* омогућава креирање функција. Могуће је специфицирати листу параметара, као повратну вредност. Синтакса креирања функције дата је на наредном листингу.

```
1     function Suma(prvi: Int, drugi: Int): Int {
2         return prvi + drugi;
3     }
```

Листинг 4. - *Функција у програмском језику Q#*

Веома сличан концепт концепту функције јесу операције. Операције представљају исто што и функције уз разлику да операције укључују логику која је везана за квантно рачунарство. Код функција, уколико проследимо исте параметре на улазу функције, она увек треба да врати исти резултат. Са друге стране, због недетерминистичке природе квантних система, операције не морају нужно да

врате исту вредност када су прослеђени исти параметри. Важно је напоменути да због своје детерминистичке природе, функције не смеју да позивају операције. Са друге стране, операције смеју да позивају функције. Пример операције је да

```
operation InicijalizujIIzmeri(): Result {
    use qubit = Qubit();
    return M(qubit);
}
```

Листинг 5. – *Пример операције у програмском језику Q#*

У претходном листингу се појављује неколико концепата који су карактеристични само за квантно програмирање. Коришћењем резервисане речи **use** алоцира се један кјубит. Синтакса програмског језика *Q#* омогућава да се одједном алоцира читав регистар кјубита тако што се поред резервисане речи **Qubit** уместо обичних загада користе угласте заграде **[ ]** у оквиру којих се специфицира број кјубита у регистру.

Још једна ствар која је карактеристична за операције и за квантни део програмског језика *Q#* је та да се као повратна вредност операције користе две класе: *Unit* и *Result*. *Unit* повратни тип представља еквивалент класичном *void* и означава да функција нема повратну вредност. Са друге стране, када је повратни тип дефинисан као *Result*, он означава резултат квантног мерења.

- Када је реч о логичким колима, односно операцијама које могу да се изврше над једним кјубитом, *Q#* омогућава да се оператори примене веома једноставно. У наредном листингу је приказан пример примене Хадамард оператора над алоцираним кјубитом.

```
operation PrimeniHadamard (): Unit {
 using (qubit = Qubit()) {
     H(qubit);

     // …

     Reset(qubit);
 }
}
```

Листинг 6. – *Примена Хадамард оператора у програмском језику Q#*

У претходном примеру видимо алтернативну синтаксу за алоцирање кјубита помоћу **using** резервисане речи. Оваква конструкција се користи и у *C#* програмском језику и служи да деалоцира ресурсе након извршавања блока кода у оквиру **using** израза.

- У случају примене оператора над више кјубита, као што је случај код *CCNOT* оператора, синтакса је веома слична претходно приказаној. У листингу који следи, приказана је алокација два контролна кјубита и једног циљног, а потом и синтакса примене *CCNOT* оператора.

```
1     operation PrimeniCCNOT () : Unit {
2      using (kontolni1 = Qubit()) {
3          using (kontrolni2 = Qubit()) {
4              using (ciljni = Qubit()) {
5                  CCNOT(kontrolni1, kontrolni2, ciljni);
6
7                  // …
8              }
9          }
10     }
11 }
```

Листинг 7. - *Примена CCNOT оператора у програмском језику Q#*

- Операцијама могу да се придодају декоратори као што се додају и у класичним програмским језицима. Тако на пример декоратор *EnrtyPoint* означава операцију код које започиње извршавање кода. По конвенцији, овај декоратор сме да се употреби искључиво једном у коду.

```
1   @EntryPoint()
2   operation Main (): Unit {
3     // kod
4   }
```

Листинг 8. – *Пример употребе декоратора у програмском језику Q#*

- Напреди концепт у програмском језику *Q#* представљају адјунговане трансформације. Адјункција произвољне трансформације *U* је њена конјугована транспозиција. [33] Како је раније објашњено, квантне трансформације, односно логичка кола, можемо да посматрамо као матрице. Појам адјункције се примењује у контексту матрица и има важну улогу у квантном рачунарству. Наиме, због честе појаве у квантом програмирању да се примени операција коју означавамо са

U како би се систем превео у одрешено стање у ком би требало да се примени оператор *V*, а да се након примене операције *V* поништи примењена операција *U*, у програмском језику *Q#* је уведена наредба конјугације. На наредном листингу је приказана синтакса ове наредбе.

```
1     within {
2       // kod
3     }
4     apply {
5       // kod
6     }
```

Листинг 9. – *Синтакса оператора конјугације*

Блок кода који је омеђен кључном речју *within* представља трансформатор који ће да буде примењен пре кода који се налази у оквиру *apply* блока. Уједно, адјункција трансформатора који се налази унутар *within* блока ће да буде примењена након извршавања трансформатора унутра *apply* блока.

- Да би за операцију коју смо написали могла да буде креирана и извршена њена адјункција, програмски језик *Q#* уводи могућност дефинисања карактеристика операција. Кључна реч за дефинисање карактеристике функције је *is* након које може да се наведе тип специјализације. Уколико желимо да *Q#* аутоматски генерише дисјункцију операције, примењујемо кључну реч *Adj*. На наредном листингу је приказано дефинисање специјализације адјункције.

```
1   operation Transformacija (): Unit is Adj {
2     // code
3   }
```

Листинг 10. – *Приказ синтаксе за дефинисање специјализације адјункције*

- Последња функционалност програмског језика *Q#* коју уводимо у оквиру ове секције јесу парцијалне функције. Ове функције омогућавају динамичку трансформацију операције на начин да креирају нову операцију која прима мање параметара. Другим речима, ако имамо функцију која прима *n* аргумената, позивом функције са *m* параметара постижемо то да се креира нова функција. Новокреираној функцији, приликом њеног позива, потребно је да се проследе

параметри који нису прослеђени иницијалној функцији од које је она настала. Ако посматрамо функцију:

function DopuniElemente (ukupnoElemenata : Int, podrazumevaniElement : 'T, ulazniNiz: 'T[]) : 'T[]

која прима три аргумента и која треба да омогући да се за прослеђени низ и број допуни број елемената тако да новонастали низ има дефинисану дужину. Пример функционисања парцијалне фунције дат је на листингу који следи.

```
1  operation Transformation (): Unit is Adj {
2    let niz = [1,2,3,4,5];
3    let dopuniNulama = DopuniElemente(_, 0, _);
4    let dopunjeniNiz = dopuniNulama(7, niz);
5    // [0, 0, 1, 2, 3, 4, 5]
6  }
```

Листинг 11. – *Приказ парцијалне функције у програмској језику* Q#

4. Циљеви истраживања

У овој секцији, фокус је стављен на прецизно дефинисање циљева истраживања. Како би систематичност и објективност у постављању и постизању циљева била задржана, примењен је *GQM* (енг. *Goal-Question-Metric*) приступ [22]. Овај приступ дефинише структуру и оквир за постављање јасних циљева, формулисање релевантних питања и идентификацију одговарајућих метрика за мерење перформанси.

4.1. Циљ истраживања

Циљ истраживања спроведеног у овом раду јесте анализа коришћења конвенционалног процеса развоја софтвера у контексту квантног софтверског инжењерства како би се показало да ли је конвенционални процес развоја софтвера применљив у квантном софтверском инжењерству, узимајући у обзир перспективу софтверског инжењера са вишегодишњим искуством у раду у контексту софтверског инжењерства.

Табела 3. *- Формулација циља истраживања коришћењем GQM шаблона*

Анализирати	Квантну технологију
У сврху	Разумевања
С обзиром на	Употребљивости у софтверском инжењерству
Из перспективе	Софтверских инжењера
У контексту	Софтверских пројеката

4.2. Истраживачка питања

Како бисмо постигли дефинисани циљ, потребно је да поставимо кључна питања:

- **(П1)** Да ли искуство у раду у контексту класичног софтверског инжењерства употребљиво или применљиво у контексту квантног софтверског инжењерства?
- **(П2)** Да ли постоје ограничења када се модел развоја софтвера из класичног софтверског инжењерства примени у контексту квантног софтверског инжењерства?

На основу досадашњих истраживања (види Поглавље 3.) није могуће формулисати једнообразни одговор на постављена питања П1 и П2. Из разлога што се тренутно већина истраживања фокусира на решавање хардверских изазова. Упркос томе, постоје озбиљне иницијативе, као што је Талавера манифест, које указују да је ова тема формулисана нашим истраживачким питањима је релевантна.

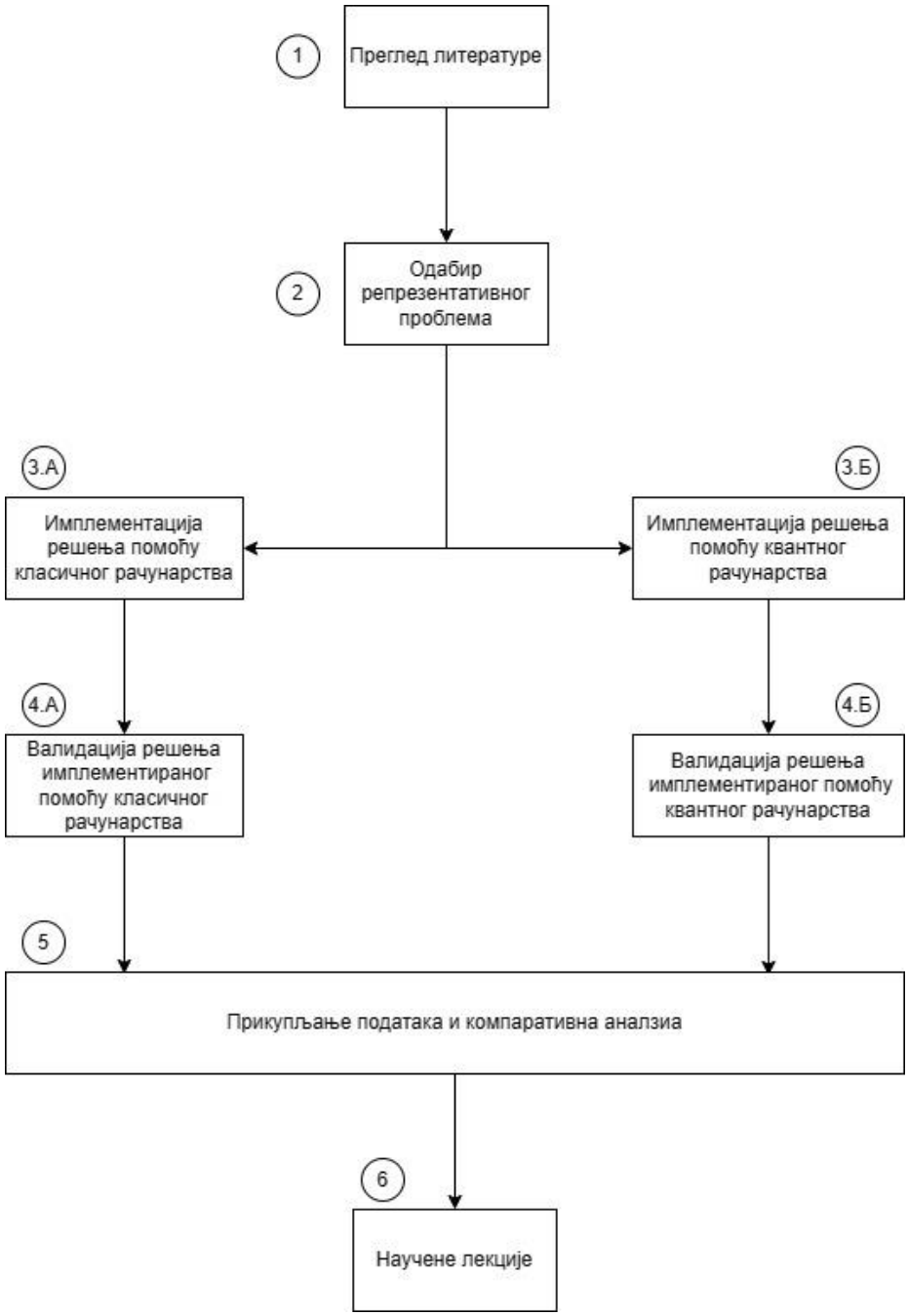
5. Дизајн студије

Како би одговори на истраживачка питања били дати и циљ истраживања постигнут, било је неопходно да се спроведе истраживање које би дало одговоре на постављена питања. У овом поглављу је детаљно описан дизајн спроведеног истраживања.

Квантно софтверско инжењерство представља област која још увек није широко распрострањена, односно још увек нема интензивну примену у индустрији развоја софтверских производа. Ова чињеница указује на то да није изводљиво извршити истраживање над групом људи која професионално ради у овој области. Са друге стране, иако постоје напори да се корисницима омогући приступ квантним рачунарима са што бољим перформансама, чињеница да постоји ограничење ресурса демотивише многе инжењере да се озбиљније посвете истраживању квантног рачунарства. Узимајући претходне чињенице у обзир, студија описана у овом раду је спроведена тако што је аутор рада био тај који је спровео истраживање. Битно је да се напомене да је аутор софтверски инжењер са вишегодишњим искуством у раду у контексту класичног софтверског инжењерства и да је његово искуство у раду са квантним технологијама на почетном нивоу.

Циљ истраживања, дефинисан у претходном поглављу, настоји да покаже да ли је процес развоја софтвера у класичном рачунарству применљив у контексту квантног рачунарства. Да бисмо постигли дефинисани циљ, студију смо дизајнирали тако да она представља компаративну студију случаја. Компаративна студија случаја је методолошки приступ у истраживању који се користи за поређење два или више

појединачна случаја ради дубљег разумевања њихових карактеристика, понашања, перформанси или утицаја. [30] Овај приступ омогућава истраживачима да истраже сличности, разлике, предности и недостатке између различитих случајева како би добили бољи увид у појаве или феномене које описују.



Слика 17. – *Дизајн студије*

Како бисмо применили компаративну анализу случаја, за поређење смо узели процес развоја софтвера у класичном рачунарству и процес развоја софтвера у квантном рачунарству. Низ корака који треба да буду спроведени у току истраживања су приказани на слици **Слика *17***.

Први корак у спровођењу истраживања подразумева *преглед литературе*. Овај корак омогућава истраживачу да се упозна са контекстом и да прикупи довољно информација за спровођење осталих корака. Након прегледа литературе, потребно је да се *одабере репрезентативан проблем*, који је решив помоћу обе врсте рачунарства. Након дефинисања проблема, приступа се његовом решавању. Важно је напоменути да процес решавања проблема мора да буде структуриран, односно пошто је циљ студије да покаже применљивост класичног процеса развоја софтвера у контексту квантног рачунарства, процес решавања проблема би требало да буде неки модел развоја процеса у квантном рачунарству. У секцији 3.4 је представљен је репрезентативан модел развоја софтвера у класичном рачунарству, тако да ће исти модел да буде употребљен у спроведеној студији.

Решење проблема треба да буде имплементирано на два начина: *имплементација помоћу класичног рачунарства* и *имплементација помоћу квантног рачунарства*. Спровођење наведених корака укључује да испитаник коришћењем класичног рачунарства, односно програмских језика, радних оквира и доступних софтверских алата направи софтверски производ који би требало да реши дефинисани проблем. Јако је важно да процес развоја прати модел развоја описан у поглављу 3.4. Итерације кроз фазе развоја процеса су дозвољене, будући да итерације представљају природан ток развоја софтверског производа.

Након што су решења имплементирана, наредни корак је *валидација креираних решења*. Као и током развоја, коришћењем доступних алата треба проверити да ли оба решења функционишу тако да задовољавају постављене захтеве. Након валидације решења, врши се прикупљање квалитативних података тако што испитаник дели искуство о томе са којим се изазовима испитаник срео током примене класичног процеса развоја софтвера у контексту квантног софтверског инжењерства. Добијени подаци се обрађују и структурирају, тако да након овог корака настаје низ *научених лекција* које јасно указују на изазове са којима се програмери сусрећу када примењују класични модел развоја софтвера у квантном софтверском инжењерству.

Како би компаративна анализа могла да буде спроведена, битно је да иста особа ради на оба решења. На овај начин, испитаник би, на основу свог искуства, требало да

буде у могућности да изнесе квалитативне чињенице о изазовима са којима се сусрео током рада на решавању проблема. На основу добијеног искуства, биће дати одговори на истраживачка питања и остварен циљ спроведене студије.

6. Резултати

На основу дизајна студије представљеног у поглављу 5, спроведена је компаративна студија случаја. У првом делу студије, проблем је решен коришћењем класичног софтверског инжењерства. У другом делу, исти проблем је решен коришћењем квантног софтверског инжењерства. Оба дела су пратила спецификацију процеса развоја софтвера. У овом поглављу приказано је како су спроведена оба дела студије.

6.1. Одабир релевантног проблема

Предуслов за одабир проблема је био да проблем може да буде решив помоћу обе врсте рачунарства. Одабиру релевантног проблема претходио је преглед литературе описан у поглављу 3. На основу прегледа литературе изабран је проблем претраге неструктурираних података. Повод за одабир овог алгоритма је била чињеница да је Гроверов алгоритам представља један од најпопуларнијих квантних алгоритама и да решава проблем претраге неструктурираних података. Уједно, узевши у обзир да је проблем претраге је изузетно важан проблем у класичном рачунарству [36], проблем претраге неструктурираних података је одабран као релевантни проблем.

6.2. Имплементација

У овој секцији, дат је опис имплементације решења помоћу класичног и квантног рачунарства. Посматране су фазе развоја које претходе фази тестирања решења.

6.2.1. Имплементација коришћењем класичног софтверског инжењерства

Према специфицираном моделу процеса развоја софтверског производа, прва фаза је фаза спецификације захтева решења. У овој студији, решење треба да омогући проналажење елемента у колекцији елемената који нису структурирани ни на који начин. За разлику од многих ситуација које су својствене пројектима у реалном свету, у којима је неопходна детаљна анализа захтева решења и њихово превођење у скуп функционалности које софтверски производ треба да задовољи, у овом случају је функционалност софтвера која треба да буде имплементирана потпуно јасна. Због поједностављења процеса развоја, решење није требало да садржи никакав кориснички интерфејс, већ је алгоритамски требало да буде решен постављени проблем.

Након спецификације захтева, наредна фаза је фаза дизајна решења. У оквиру ове фазе су донете одлуке која технологија ће да се користи за имплементацију решења, које развојно окружење ће да буде употребљено и који приступи ће да буду примењени за валидацију решења. Програмски језик одабран за имплементацију овог решења јесте *C#*. Овај програмски језик представља један од најпопуларнијих програмских језика у тренутку писања овог рада. Популарност језика је заснована на његовим перформансама и дизајну који програмерима омогућава писање кода који може да се извршава на великом броју хардвера. Развојно окружење које је коришћено јесте Вижуал студио (енг. *Visual Studio*). Ово окружење пружа изузетно корисничко искуство приликом писања софтвера коришћењем програмског језика *C#*. Захваљујући поменутој чињеници да кориснички интерфејс није неопходан, тип апликације који је креиран је конзолна апликација (енг. *Console application*). Апликација је осмишљена тако да се у оквиру изворног кода иницијализује колекција података и да се такође одабере један елемент који ће да буде предмет претраге. Требало би да буде креирана функција која ће као аргументе да има претходно поменуту колекцију података и податак који треба да се тражи у колекцији. Резултат функције биће потврда да ли је податак пронађен или није.

Када је решење концептуално дизајнирано, приступа се процесу имплементације решења. Овај корак најчешће укључује покушај проналаска постојећег решења које би могло да се искористи. Будући да је проблем претраге један од најосновнијих проблема у програмирању, ауторово искуство је било довољно како би проблем био решен. Наиме, будући да се ради о неструктурираној колекцији података, решење је имплементирано тако да се пролази кроз елементе колекције редом и проверава да ли је вредност у некој од итерација једнака траженој вредности. Уколико јесте, алгоритам се у том тренутку завршава и није потребно правити остале итерације. Овај алгоритам је најефикаснији у случају да се тражени елемент налази на почетку колекције, а најлошији када се елемент налази на крају колекције. У наредном листингу приказан је програмски код којим је имплементирано решење.

```
bool NadjiBroj(int[] kolekcijaPodataka, int trazeniBroj)
{
    for (int i = 0; i < kolekcijaPodataka.Length; i++)
    {
        if (kolekcijaPodataka[i] == trazeniBroj)
        {
            return true;
        }
    }

    return false;
}

int[] testKolecijaPodataka = new int[10] { 5, 2, 66, 12, 24, 11,
7, 35, 81, 44 };

int testTrazeniBroj = 15;

Console.WriteLine(NadjiBroj(testKolecijaPodataka,
testTrazeniBroj));
```

Листинг 12. – *Имплементација решења коришћењем класичног софтверског инжењерства*

Приказано решење настало је као резултат неколико итерација рефакторисања кода. У случајевима када је функционалност комплексна, имплементација захтева да се код покреће део по део како би програмер знао да ли развој иде у правом смеру. У овој студији, итерације су служиле да се код форматира на начин да буде што јаснији и ефикаснији.

Са фазом имплементације се константно смењује фаза дебаговања, на начин да након што програмер напише код и крене да га проверава, наиђе на нелогичности које проверава тако што извршава линију по линију кода. Програмски језик *C#* и Вижуал студио омогућавају извршавање дебаговања. На овај начин, програмер проверава вредности променљивих у меморији у сваком тренутку и на тај начин идентификује грешке много лакше него када би требало закључке да доноси на нивоу читавог решења. Након неколико итерација, апликација је спремна за прелазак у фазу тестирања.

6.2.2. Имплементација коришћењем квантног софтверског инжењерства

Као и током имплементације описане у претходном одељку, и ово решење је имплементирано праћењем дефинисаног модела развоја софтверског производа. У оквиру прве фазе, фазе спецификације захтева решења, утврђен је скуп функционалност које апликација треба да испуни. Као што је објашњено у оквиру првог дела студије, због јасноће дефинисаног проблема, било је једноставно утврдити да је функционалност коју софтвер треба да имплементира проналажење постојања елемента у неструктурираној колекцији података. Комплексност спровођења задатка ове фазе не треба потцењивати будући да квантно рачунарство, према досадашњим подацима, у могућности да реши одређене класе проблема. Стога би, у случајевима где треба имплементирати сложене функционалности, било тешко да се захтеви корисника преведу у могућности које пружа квантно рачунарство.

Наредна фаза, фаза дизајна решења, служи да се решење дефинише на концептуалном нивоу. На почетку овог задатка, будући да аутор није имао искуства у раду са квантним технологијама, требало је да се истражи да ли постоји готово решење за овај проблем или део решења који би могао да се искористи. Ауторово искуство, које је подразумевало познавање основа квантног рачунарства и рада са квантним логичким колима било је довољно за покушај да се пронађе одговарајући алгоритам за овај проблем. Након утрошеног времена на истраживање литературе и искустава других програмера, одлучено је да за се за решавање овог проблема искористи Гроверов алгоритам претраге неструктурираних база података. Дизајн овог алгоритма објашњен је у одељку 3.3. Како би особа која се бави класичним софтверским инжењерством могла

да разуме овај алгоритам, потребно је изузетно добро познавање теоријских концепата квантног рачунарства. Са друге стране, познавање теоријских концепата и логичких кола се показало као недовољан предуслов да аутор буде у могућности да сам дође до алгоритма који би решио постављени проблем. Разумевање Гроверовог алгоритма је било изузетно захтевно и тај корак је узео доста више времена него што је очекивано. Након установљавања који алгоритам се користи, требало је донети одлуку применом које технологије би решење требало да буде имплементирано. Након истраживања неколицине доступних решења, аутор је донео одлуку да решење буде имплементирано коришћењем програмског језика *Q#*. Овај језик је изабран, пре свега, због могућности да се извршава на симулатору који може да буде покренут на локалном рачунару. Поред тога, типизација података коју конкурентни Кискит не поседује је још један од разлога зашто је овај програмски језик изабран. За развојно окружење је изабран Вижуал студио код (енг. *Visual Studio Code*). Овај алат, поред одличних перформанси и богатог корисничког искуства које пружа, поседује екстензије које олакшавају рад са програмским језиком *Q#*. Будући да Мајкрософтов пакет за развој софтвера нуди неколико могућих типова апликација, за потребе овог пројекта је изабрана конзолна апликација. Након доношења претходно описаних одлука, завршен је процес дизајна решења и прелази се на фазу имплементације.

Задатак у фази имплементације био је да се имплементира Гроверов алгоритам. Први покушај имплементације се заснивао на употреби ауторовог предзнања о квантном програмирању и праћењу дизајна система дефинисаном у претходном кораку. Због сложености Гроверовог алгоритма, било је неопходно осврнути се на фазу дизајна пуно пута. Први корак алгоритма, односно превођење свих кјубита у стање суперпозиције уз помоћ Хадамард оператора је био урађен без већих проблема. Наредни корак, креирање црне кутије за обележавање траженог стања је представљао велики изазов. Пре свега, изазов је представљало писање кода које одговара овако комплексној компоненти која представља непознаницу, будући да се овакве компоненте не креирају у класичном софтверском инжењерству. У првој итерацији, написан је код за који је аутор сматрао да би могао да буде валидно решење. Како би се потврдило да ли неки део ради како треба, потребно је покренути програм и погледати излазе који се добијају. Због природе кјубита, постоји велико ограничење због ког није могуће измерити вредности јер се у том случају стање кјубита одмах уништава и оно постаје 0 или 1. Исто тако, сама природа програма написаном коришћењем квантног рачунарства подразумева да систем ради са континуалним величинама, не са дискретним, што онемогућава писање програма у

итерацијама. Читав програм представља механизам који је спрегнут од почетка до краја и писање компоненти независно једне од друге не даје праву слику о томе како би систем требало да функционише. Ова чињеница представља велики изазов, будући да се програми у класичном рачунарству пишу инкрементално, део по део.

Како писање спонтаног решења није био успешан приступ, други покушај је подразумевао покушај да се пронађе постојеће решење које би могло да се искористи за решење постојећег проблема. Како је Гроверов алгоритам један од најпопуларнијих алгоритама у квантном програмирању, ресурси доступни на интернету су разноврсни и могуће је пронаћи различите имплементације Гроверовог алгоритма. Након детаљне анализе доступних решења, на наредном листингу је приказана имплементација Гроверовог алгоритма искоришћена за потребе ове студије.

```
1     namespace GroverovAlgoritam {
2
3      open Microsoft.Quantum.Canon;
4      open Microsoft.Quantum.Intrinsic;
5      open Microsoft.Quantum.Measurement;
6      open Microsoft.Quantum.Math;
7      open Microsoft.Quantum.Convert;
8      open Microsoft.Quantum.Arrays;
9      open Microsoft.Quantum.Arithmetic;
10
11     operation CrnaKutija(trazenaVrednost : Int, qubits :
12 Qubit[]): Unit is Adj {
13
14         let n = Length(qubits);
15         within {
16             let markerBits = IntAsBoolArray(trazenaVrednost, n);
17             for i in 0..n-1 {
18                 if not markerBits[i] {
19                     X(qubits[i]);
20                 }
21             }
22         } apply {
23             Controlled Z(Most(qubits), Tail(qubits));
24         }
25     }
26
27     function PripremiCrnuKutiju(trazenaVrednost : Int) :
28 ((Qubit[]) => Unit is Adj) {
29         return CrnaKutija(trazenaVrednost, _);
30     }
31
32     operation Difuzija(qubits : Qubit[]) : Unit is Adj {
33         within {
34             ApplyToEachA(H, qubits);
35             ApplyToEachA(X, qubits);
```

```
36          } apply {
37              Controlled Z(Most(qubits), Tail(qubits));
38          }
39      }
40
41      operation Grover(n : Int, crnaKutija : (Qubit[]) => Unit is
42  Adj) : Int {
43          let brojIteracija = Floor(PI() / 4.0 * Sqrt(IntAsDouble(2
44  ^ n)));
45          use qubits = Qubit[n];
46          ApplyToEach(H, qubits);
47
48          for x in 1 .. brojIteracija {
49              crnaKutija(qubits);
50              Difuzija(qubits);
51          }
52
53          let registar = LittleEndian(qubits);
54          let broj = MeasureInteger(registar);
55          return broj;
56      }
57
58      @EntryPoint()
59      operation Main() : Unit {
60          let brojKjubitaURegistru = 3;
61          let trazenaVrednost = 5;
62
63          let brojStanja = 2^brojKjubitaURegistru;
64          let crnaKutija = PripremiCrnuKutiju(trazenaVrednost);
65
66          let nadjenaVrednost = Grover(brojKjubitaURegistru,
67  crnaKutija);
68          Message($"Trazena vrednost: {trazenaVrednost}");
69          Message($"Nadjena vrednost: {nadjenaVrednost}");    }
70
71  }
```

Листинг 13. – *Имплементација Гроверовог алгоритма у програмском језику Q#*

Алгоритам је тако дизајниран да се дефинише број кјубита у регистру и да се специфицира тражена вредност. Имплементација црне кутије је урађена на начин да она не представља фиксни низ операција, већ се прилагођава броју кјубита и траженом стању. Дифузиони оператор је имплементиран коришћењем X логичког кола и контролисаног Z оператора. Пролази се кроз оптималан број итерација. Алгоритам након извршавања треба да врати тражену вредност.

Када је реч о фази дебаговања, спровођење ове фазе је могуће захваљујући квантним симулаторима. На правом квантном хардверу, није могуће извршити никакву врсту дебаговања. Иако је на симулаторима могуће да се прекине ток програма, једина

ствар коју симулатор може да нам покаже јесте вредност стања након операције мерења. Коришћени симулатор није у могућности да прикаже међустање, односно да са прецизношћу покаже бројеве док је кјубит у суперпозицији. Немогућност дебаговања, мерења стања и развоја алгоритма инкрементално су учинили спровођење овог дела студије веома тешким.

6.3. Валидација

У овој секцији, приказано је на који начин су валидирана решења чији је процес имплементације описан у претходном одељку.

6.3.1. Валидација решења имплементираног помоћу класичног рачунарства

Након завршене имплементације, на реду је фаза тестирања апликације. Тестирање представља фазу у којој би требало да се потврди да апликација има имплементиране све тражене функционалности и да те функционалности раде тачно онако како је специфицирано у првој фази процеса развоја. Тестирање апликације може да се уради на два начина: ручно или аутоматски. У овој студији, спроведене су обе врсте тестирања. Када је реч о ручном тестирању, неопходно је да оно буде одрађено од стране особе која добро познаје то како би апликација требало да функционише и да на основу коришћења апликације утврди да ли све ради као што је очекивано. Уколико је све у реду, апликација се сматра валидном и прелази у фазу постављања у продукционо окружење. У супротном, уколико је било какав проблем идентификован, процес развоја се враћа у фазу имплементације где програмер настоји да на основу описа проблема отклони грешку. У овим ситуацијама, дебаговање представља алат од велике користи. За разлику од ручног тестирања, постоји и могућност да се тестови извршавају аутоматски. Аутоматизовано тестирање се извршава тако што је написан програмски код који ће уместо ручног тестирања да све кораке одради аутоматски на основу кода којим је написан. Овај приступ омогућава постизање веће поузданости система и представља доста ефикасније решење када је тестирање у питању. Пример једног аутоматизованог теста који је написан у оквиру ове студије случаја је дат на наредном листингу.

```
[TestMethod]
public void NadjiBroj_ShouldReturnTrue_WhenNumberExists()
    {
        // Arrange
        int[] testKolekcijaPodataka = new int[10] { 5, 2, 66, 12,
24, 11, 7, 35, 81, 44 };
        int testTrazeniBroj = 12;

        // Act
        bool result = NadjiBroj(testKolekcijaPodataka,
testTrazeniBroj);

        // Assert
        Assert.IsTrue(result);
    }
```

Листинг 14. – *пример аутоматизованог теста за решење имплементираног класичним софтверским инжењерством*

Након што је валидност апликације потврђена, апликацију је потребно поставити на хардвер на ком ће апликација да се извршава и где ће крајњи корисници да га користе. Апликација креирана у овој студији не представља апликацију која би могла да буде коришћена од стране великог броја корисника, будући да је изостављен графички кориснички интерфејс. Са друге стране, у случају креирања апликација које треба да буду коришћене од стране великог броја корисника, потребно је добро анализирати све могућности и одабрати платформу на којој ће хардвер моћи да подржи све захтеве које апликација са собом носи. Са појавом клауд технологија, програмерима је омогућено да платформе бирају тако што могу да конфигуришу већину параметара, стога највећи изазова представља разумевање могућности доступних решења и одабир најприкладнијег за креиран софтверски производ.

Након што је апликација постављена на продукционо окружење, сматрамо да је развој софтверског производа завршен и апликација прелази у процес одржавања.

6.3.2. Валидација решења имплементираног помоћу квантног рачунарства

У тренутку када је програм завршен са становишта фаза имплементације и дебаговања, по дефинисаном моделу развоја софтвера уследила је фаза тестирања решења. У првој итерацији тестирања, програм је тестиран мануелно, тако што је провераван испис програма. Пошто је утврђено да програм функционише како треба, наредни корак је био разматрање имплементације аутоматизованих тестова. Иако у иницијалном дизајну система није специфицирано писање аутоматизованих тестова, од значаја за ову студију је утврђивање како се писање аутоматских тестова у квантном софтверском инжењерству утиче на читав развојни процес и у којој мери уводи изазове за софтверске инжењере. Током писања аутоматских тестова за имплементирани алгоритам, јавили су се одређени технички проблеми. Наиме, у тренутној верзији коришћеног квантног софтверског пакета није омогућено тестирање на начин да се валидирају добијена и очекивана вредност. Будући да искуство из класичног софтверског инжењерства у овој фази није било од помоћи, у оквиру ове студије нису имплементирани аутоматизовани тестови. Током спровођења корака за имплементацију аутоматизованих тестова, уочена је још једна отежавајућа околност. Наиме, због природе квантних система где они функционишу као целина, није могуће да се врши тестирање појединачних компоненти у изолацији. Ова чињеница представља отежавајућу околност у процесу тестирања јер је јако важно да се утврди да је сваки део алгоритма или неког комплексног система исправан како бисмо знали да је и читав систем исправан. С обзиром на то да аутоматски тестови нису креирани, решење је валидирано мануелном провером излазних вредности.

Последња фаза је фаза у којој се доноси одлука где ће продукциона инстанца апликације да буде смештена и покренута. Будући да су претходни кораци извршавани на ауторовом локалном рачунару, који у себи нема квантни хардвер, било је неопходно да се написани код постави на квантни хардвер. Као што је дискутовано и у првом делу студије, доношење ове одлуке захтева детаљну анализу доступних могућности и доношења одлуке која опција не најбоља. Узимајући у обзир да је компанија Мајкрософт аутор програмског језика *Q#* и да уједно нуди могућност извршавања софтвера на квантном у оквиру платформе Ежр, ова опција би била најбоље решење. Због ограничења

ресурса у оквиру ове студије, овај корак није спроведен и стога у наставку није дато више детаља.

Спровођењем свих корака које предвиђа одабрани модел процеса развоја софтвера, дошло се до производа који представља решење постављених проблема. Упркос бројним изазовима који су се појављивали у току рада, студија је завршена тако што су креирана два софтверска производа, која су имплементирана коришћењем различитих врста рачунарства и која решавају постављени проблем.

6.4. Прикупљање података и компаративна анализа

Методологија примењена у студији описаној у овом раду је компаративна студија случаја развоја софтверског производа коришћењем класичног и квантног софтверског инжењерства у контексту модела процеса развоја софтвера прихваћеног у класичном софтверском инжењерству. Резултат примене ове методологије је новонастало искуство испитаника о сличностима и разликама примене модела процеса развоја софтвера помоћу различитих облика рачунарстава. Искуство испитаника треба структурирати како би се добили подаци који би дали одговор на постављена истраживачка питања.

Начин који је примењен за структурирање искуства испитаника је креирање скупа научених лекција. Научене лекције представљају исказе који сведоче о изазовима примене модела класичног процеса развоја софтвера на квантно софтверско инжењерство. Будући да научене лекције представљају квалитативне податке, није могуће одредити границу која би представљала довољан да сматрамо да је класичан процес развоја софтвера тешко применљив у контексту квантног рачунарства. У наредном поглављу су приказане научене лекције.

Узимајући у обзир да се модел процеса развоја софтвера састоји од неколико фаза, од испитаника је захтевано да за сваку фазу изнесе макар једну научену лекцију. Подела научених лекција по фазама би требало да да одговор на то да су можда само неке од фаза тешко применљиве у контексту квантног рачунарства.

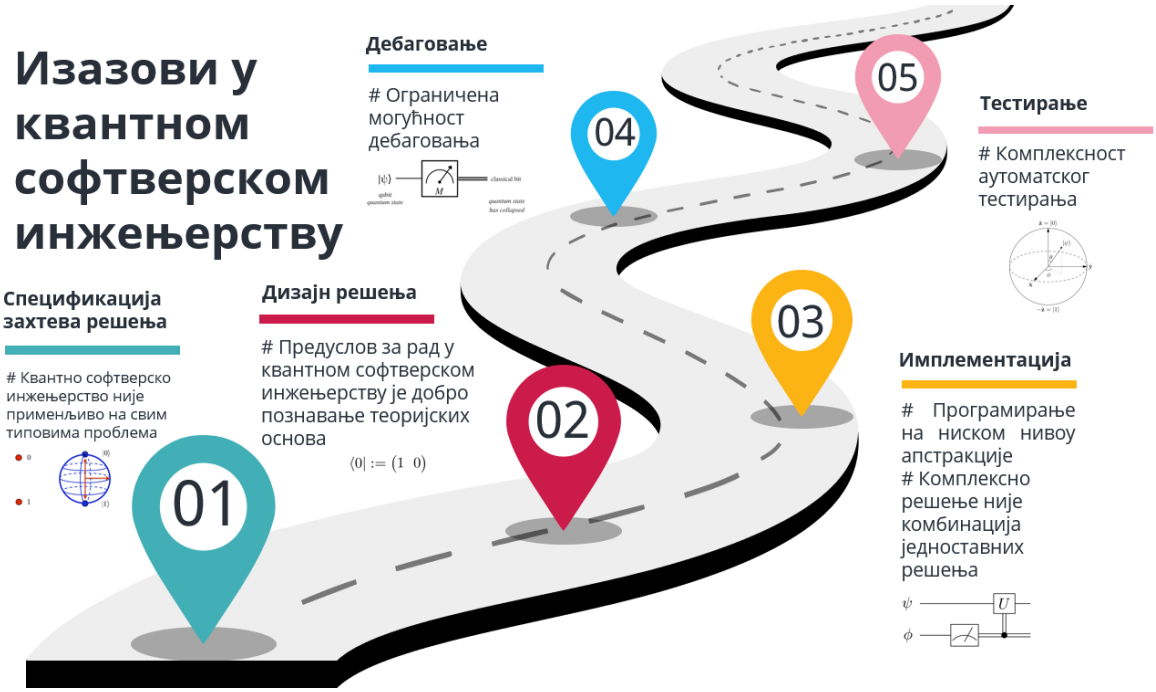
6.5. Валидност резултата

Свако истраживање се спроводи над одређеном популацијом и величина и разноврсност те популације утичу на одлуку колико истражену појаву и добијене резултате можемо да генерализујемо и у којој мери можемо да тврдимо да они важе. У оквиру овог рада, спроведена је компаративна студија случаја. Будући да се ради о студији случаја, знамо да је истраживање спроведено на једном конкретном случају и не може се говорити о закључцима који могу да се генерализују и важе за све остале случајеве. Са друге стране, будући да је аспект квантног рачунарства који је посматран у овој студији још увек недовољно истражен, резултати овог рада могу да буду од користи другим радовима, као и људима који се налазе у неком сличном случају у односу на случај истражен у раду.

Још једна важна чињеница која треба да се узме у обзир јесте да се аутор рада тај који је спровео истраживање. Ауторово искуство у развоју софтвера у класичном рачунарству траје више од три године. Квантним рачунарством је крену да се бави пре мање од 6 месеци и самим тим представља доброг кандидата који би могао да стекне утисак о томе какви се изазови појављују у квантном софтверском инжењерству а да нису примећени у класичном софтверском инжењерству. Такође, аутор је имао прилике да се сусретне са теоријом квантне физике и да разуме одређење делове физичког аспекта квантног рачунарства. Чињеница да је аутор рада једини спровео истраживање поново умањује могућност да се решење генерализује, али може да буде од користи програмерима који желе да се опробају у светку квантног рачунарства, а који долазе са претходним искуством из класичног софтверског инжењерства.

7. Научене лекције

Након спровођења истраживања, наредни задатак је био да се прикупљено искуство структурира и преточи у низ научених лекција. На основу научених лекција, требало би да се дају одговори на постављена истраживачка питања, чиме би био постигнут циљ ове студије. На слици **Слика *18***. дат је приказ научених лекција у спроведеном истраживању.



Слика 18. - *Инфографик изазова у квантом софтверском инжењерству*

(1) **Квантно рачунарство не може да реши све врсте проблема.** Квантно рачунарство је млада област и у току њене историје, откривено је да само одређене класе проблема могу да буду решене коришћењем ове технологије. Са друге стране, постоје проблеми који могу да се реше коришћењем квантног рачунарства, али таква решења показују лошије перформансе у односу на решења имплеметирана помоћу класичног рачунарства. Стога је у фази спецификације захтева решења од велике важности да се утврди да ли је проблем решив коришћењем квантне технологије и уколико јесте, да ли су перформансе тог решења боље од неког конвенционалног решења. За разлику од класичног рачунарства, где у фази спецификације захтева могу да учествују нетехничка лица, у случају квантног рачунарства је неопходно добро познавање могућности квантног рачунарства.

(2) **Програмирање на ниском нивоу апстракције**. Класично софтверско инжењерство је достигло ниво развоја где је програмерима хардвер потпуно апстрахован и где значајан број људи који користе ову технологију немају свест о томе шта се дешава на хардверском нивоу. Програмски језици сакривају све детаље имплементације и главни задатак програмера је да напишу логику апликације која ће да одговара потребама крајњег купца. У поређењу са оваквом парадигмом, прелазак на квантно програмирање представља велику промену, будући да квантно програмирање представља програмирање на доста нижем нивоу апстракције. Програмирање логичких кола у квантном рачунарству представља аспект са којим аутор није имао прилике да се сусретне у току своје професионалне каријере стога је прилагођавање на такав начин размишљања представљао велики изазов. Овај изазов би могао да представља разлог због ког би многи програмери одустали у покушају да се пребаце на квантну технологију. Са друге стране, иако овај вид програмирања представља програмирање на нижем нивоу у односу на оно што се примењује у класичном софтверском инжењерству, постоји реално очекивање да ће временом и квантно програмирање да достигне довољан ниво апстракције како би постало примамљивије програмерима који започињу рад са овом технологијом.

(3) **Сложено решење није скуп једноставних решења.** Природа квантних алгоритама која је дискутована у више наврата у оквиру овог рада представља велики изазов за инжењере. Наиме, перцепција система као складне целине а не као скупа компоненти представља потпуно другачију перцепцију у односну на

оно што класично софтверско инжењерство познаје. У складу са тим, да би програмер радио на неком квантном алгоритму, није довољно да комбиновањем познатих концепата и логичких кола добије алгоритам који ради одређени задатак. Овај изазов, поред проблема на концептуалном нивоу, показало се да представља и проблем на нивоу имплементације. Реч је о томе да током имплементације решења није могуће покретати програм с времена на време и валидирати претходно урађено. Компонента решења нема смисла уколико није део читавог решења. Читав алгоритам представља целину која једино може да се декомпонује на концептуалном нивоу, али у току извршавања, подела није могућа.

(4) **Познавање теоријских концепата је предуслов за рад са квантним рачунарством.** За разлику од класичног софтверског инжењерства, где је ниво апстракције програмских језика достигао веома висок ниво, велики програмера може да ради без икаквог познавања теоријских концепата логичких кола и других компоненти на нижем нивоу апстракције. Са друге стране, квантни алгоритам мора да буде поткрепљен теоријским основама, помоћу којих се показује да алгоритам, уколико је исправно имплементиран, може да постигне боље перформансе од класичног алгоритма. Ослањајући се на дефинисане теоријске основе, алгоритам може да се имплементира помоћу одговарајућег програмског језика. Постојање јаке везе између теоријских концепата и имплементације представља још један од разлога за потенцијално неприхватање ове технологије од стране програмера из света класичног рачунарства.

(5) **Ограничена могућност дебаговања.** Пролазак кроз код линију по линију и посматрање промена вредности променљивих представља кључну активност у отклањању грешака и бољем разумевању програмског кода. Квантно рачунарство, нажалост, програмере оставља без овакве могућности. Временом, појавили су се дебагери који могу да функционишу једино када се ради на квантним симулаторима. Ипак, и овакав вид дебаговања не омогућава програмерима исто корисничко искуство. Наиме, будући да је стање кјубита у суперпозицији нека вредност из континуалног скупа вредности, ако бисмо хтели да знамо која је то вредност, морали бисмо да применимо операцију мерења. Уколико се мерење спроведе, стање кјубита узима вредност 0 или 1, чиме губимо информацију о томе колико је било стање у суперпозицији. Сама чињеница да од дебагера можемо само да добијем податак да ли је у неком од међукорака била

нула или јединица у многим случајевима не представља информацију од великог значаја. Овакво стање ствари представља јако велики изазов и фундаментално мења поглед да програмирање код инжењера који раде са класичним рачунарством.

(6) Комплексност имплементације аутоматског тестирања. Писањем аутоматских тестова у класичном рачунарству, могуће је тестирати како систем функционише на различитим нивоима. На пример, можемо да тестирамо како систем функционише као целина, затим како компоненте система интерагују једна са другом, али и како компоненте функционишу у изолацији. Искуство при раду на имплементацији решења коришћењем квантне технологије је показало да није могуће имплементирати све претходно поменуте аутоматске тестове у квантном софтверском инжењерству. Наиме, реч је о томе да је током тестирања Гроверовог алгоритма једино било могуће да се тестира алгоритам као целина. Иако се алгоритам састоји од јасно дефинисаних целина на концептуалном нивоу, писање аутоматских тестова за компоненте у изолацији није могуће јер алгоритам једино може да функционише као целина, не као скуп независних и појединачних компоненти.

На основу знања прикупљеног током спровођења ове студије, формулисани су одговори на постављена истраживачка питања.

- **(П1)** Искуство стечено током година рада на пројектима који користе класично софтверско инжењерство олакшава рад у квантном софтверском инжењерству. Наиме, иако квантно рачунарство уводи нове концепте, у основи, ове врсте рачунарства деле заједничку ствар – реч рачунарство. Иако се у квантном рачунарству користи кјубит као основна јединица информације, у класичном рачунарству такође имамо основну јединицу информације, само са другачијим физичким карактеристикама. Ако се померимо за степен апстракције више, долазимо до концепта логичких кола, која се појављују у обе врсте рачунарства. Особа која познаје концепте класичног рачунарства може веома брзо да разуме овај аспект квантног рачунарства. Ако са друге стране посматрамо концепте на нивоу програмских језика, и ту сусрећемо доста повезаности између језика у две различите врсте рачунарства. Функције, повратне вредности, аргументи и промењљиве представљају само неке од сличности на нивоу програмских језика. Недвосмислено, поред поменутих сличности постоје и велике разлике, али на

основу научених лекција можемо да констатујемо да претходни рад у контексту класичног инжењерства олакшава рад у контексту квантног софтверског инжењерства.

- **(П2)** На основу спроведеног истраживања настало је 6 лекција које показују изазове са којима се софтверски инжењер са вишегодишњим искуством у раду са класичним софтверским инжењерством сусрео током рада са квантном технологијом. Неки од изазова су се показали као изазови који настају због другачије природе квантних система и који би са временом били превазиђени. Са друге стране, током спровођења студије су се појавили и проблеми који представљају озбиљан изазов и који не могу тако лако да се превазиђу. На пример, одсуство могућности дебаговања представља потпуну промену погледа на програмирање у односу на оно што је програмерима познато у класичном рачунарству. Због оваквих изазова, можемо да констатујемо да постоје ограничења у примени модела процеса развоја софтвера из класичног рачунарства у контексту квантног софтверског инжењерства и да ће у будућности можда да се појаве модели процеса развоја који би били прилагођенији природи квантних система.

8. Закључак

Побољшања која доносе квантни алгоритми представљају доказ о томе колики потенцијал ова технологија поседује. Иако још увек квантно рачунарство није развијено до нивоа да буде широко примењено, све веће интересовање, како академске заједнице, тако и лидерских компанија у свету информационих технологија доприноси томе да се ова технологија све више развија. Хардверска ограничења и даље представљају препреку за имплементацију алгоритама као што су Гроверов и Шоров алгоритам и њихову примену ван истраживачког домена.

Поред хардверских ограничења, како би квантне технологије могле да достигну ниво комерцијалне употребе, биће потребно да се установи процес развоја софтверског производа, који ће да омогући структуриран и плански оријентисан процес креирања софтверског производа. Процес развоја софтвера, поред структуре коју обезбеђује, омогућава увођење већег броја чланова развојног тима који би своје задатке обављали по установљеном плану развојног процеса. Како процес развоја софтвера у квантном рачунарству до сада није истражен у довољној мери, овај рад је показао како изгледа процес развоја преузет из класичног софтверског инжењерства када се примени у контексту квантних технологија. Насупрот овом приступу, постоје разне иницијативе за креирање потпуно нових, боље прилагођених модела процеса развоја софтвера у контексту квантног рачунарства.

Истраживање спроведено у овом раду је показало да је модел процеса развоја софтвера у квантном рачунарству могуће применити у квантном софтверском инжењерству, али да он доноси неколицину изазова са собом. На пример, инкрементални развој система који подразумева развој појединачних компоненти, па потом њихово интегрисање, није могуће у квантном рачунарству. Такође, дебаговање програмског кода

и посматрање промене вредности променљивих корак по корак није могуће због природе кјубита. Претходно наведени примери представљају фундаменталне делове процеса развоја у класичном рачунарству и без њих, тешко је замислити софтверско инжењерство. Претходно изнети закључци представљају резултат истраживања на основу искуства једног човека. У наредним истраживањима, од великог значаја би било да већи број инжењера са различитим нивоима искуства спроведе слично истраживање како би валидност изнетих података била још већа.

На основу изнетих чињеница у овом раду можемо да закључимо да квантно рачунарство неће да замени у потпуности класично рачунарство. Велика је вероватноћа да ће ове две врсте рачунарства да коегзистирају заједно у одређеном периоду у будућности. У том контексту, један од изазова који треба решити јесте како интегрисати класично софтверско инжењерство, које је изграђено око једног модела апстракције софтвера, са квантним софтверским инжењерство, које изграђено око фундаментално другачијег модела апстракције софтвера. Овај изазов представља један праваца у ком би будућа истраживања на тему квантног софтверског инжењерства могла да се оријентишу.

Референце

[1] G. O'Regan, *A Brief History of Computing*. Cham: Springer International Publishing, 2021. doi: https://doi.org/10.1007/978-3-030-66599-9.

[2] Nouredine Zettili, *Quantum mechanics : concepts and applications*. Hoboken, Nj: Wiley, 2009.

[3] National Academies Of Sciences, Engineering, And Medicine (U.S.). Committee On Technical Assessment Of The Feasibility And Implications Of Quantum Computing, National Academies Of Sciences, Engineering, And Medicine (U.S.). Computer Science And Telecommunications Board, National Academies Of Sciences, Engineering, And Medicine (U.S.). Intelligence Community Studies Board, and National Academies Of Sciences, Engineering, And Medicine (U.S.). Division On Engineering And Physical Sciences, *Quantum computing : progress and prospects*. Washington, Dc: The National Academies Press, 2019.

[4] A. Hagar and M. Cuffaro, "Quantum Computing," *Stanford Encyclopedia of Philosophy*, Sep. 30, 2019. https://plato.stanford.edu/entries/qt-quantcomp/

[5] P. J. J. O'Malley et al., "Scalable Quantum Simulation of Molecular Energies," *Physical Review X*, vol. 6, no. 3, Jul. 2016, doi: https://doi.org/10.1103/physrevx.6.031007.

[6] "Google moves toward quantum supremacy with 72-qubit computer," *Science News*, Mar. 05, 2018. https://www.sciencenews.org/article/google-moves-toward-quantum-supremacy-72-qubit-computer

[7] “Google may have taken a step towards quantum computing ‘supremacy’ (updated),” *Engadget*. https://www.engadget.com/2019-09-23-google-quantum-supremacy.html?guccounter=1 (accessed Aug. 29, 2023).

[8] “Google conducts largest chemical simulation on a quantum computer to date,” *phys.org*. https://phys.org/news/2020-08-google-largest-chemical-simulation-quantum.html

[9] IBM, “IBM Quantum Computing | Roadmap,” *www.ibm.com*, Oct. 01, 2015. https://www.ibm.com/quantum/roadmap

[10] Venkateswaran Kasirajan, *Fundamentals of Quantum Computing*. Springer Nature, 2021.

[11] “What is a qubit? | Institute for Quantum Computing,” *uwaterloo.ca*. https://uwaterloo.ca/institute-for-quantum-computing/quantum-101/quantum-information-science-and-technology/what-qubit#superconducting (accessed Aug. 29, 2023).

[12] D. A. Ray, “7 Core Qubit Technologies for Quantum Computing,” *Amit Ray*, Dec. 10, 2018. https://amitray.com/7-core-qubit-technologies-for-quantum-computing (accessed Aug. 29, 2023).

[13] E. R. Johnston, N. Harrigan, and M. Gimeno-Segovia, *Programming Quantum Computers*. “O’Reilly Media, Inc.,” 2019.

[14] J. D. Hidary, Quantum Computing: An Applied Approach. Cham: *Springer International Publishing*, 2021. doi: https://doi.org/10.1007/978-3-030-83274-2.

[15] Filip Wojcieszyn, “Quantum Computing,” *Quantum science and technology*, pp. 89–132, Jan. 2022, doi: https://doi.org/10.1007/978-3-030-99379-5_4.

[16] “Cryptographers Take On Quantum Computers - IEEE Spectrum”, *spectrum.ieee.org*. https://spectrum.ieee.org/cryptographers-take-on-quantum-computers (accessed Aug. 29, 2023).

[17] Bradben, “The qubit in quantum computing - Azure Quantum,” *learn.microsoft.com*. https://learn.microsoft.com/en-us/azure/quantum/concepts-the-qubit

[18] M. Piattini, G. Peterssen, and R. Pérez-Castillo, “Quantum Computing: A New Software Engineering Golden Age,” *ACM SIGSOFT Software Engineering Notes*, vol. 45, no. 3, pp. 12–14, Jul. 2020, doi: https://doi.org/10.1145/3402127.3402131.

[19] M. R. El aoun, H. Li, F. Khomh, and M. Openja, “Understanding Quantum Software Engineering Challenges An Empirical Study on Stack Exchange Forums and GitHub Issues,”

IEEE Xplore, Sep. 01, 2021. https://ieeexplore.ieee.org/abstract/document/9609196/ (accessed Aug. 25, 2023).

[20] S. Ali, T. Yue, and R. Abreu, "When software engineering meets quantum computing," *Communications of the ACM*, vol. 65, no. 4, pp. 84–88, Apr. 2022, doi: https://doi.org/10.1145/3512340.

[21] "Five years ago today, we put the first quantum computer on the cloud. Here's how we did it.," *IBM Research Blog*, Feb. 09, 2021. https://research.ibm.com/blog/quantum-five-years

[22] V. Sylaidis, I. Nanakis, and V. Kopanas, "Introducing the Goal-Question-Metric approach to telecommunications software development: the PITA experiment," *Reliability, Quality and Safety of Software-Intensive Systems*, pp. 215–230, 1997, doi: https://doi.org/10.1007/978-0-387-35097-4_17.

[23] L. Bartlett and F. Vavrus, "Comparative Case Studies: An Innovative Approach," *Nordic Journal of Comparative and International Education (NJCIE)*, vol. 1, no. 1, Jul. 2017, doi: https://doi.org/10.7577/njcie.1929.

[24] "Quantum computing: the hype and hopes | ABB," *News*, Feb. 23, 2021. https://new.abb.com/news/detail/74736/quantum-computing-the-hype-and-hopes (accessed Aug. 29, 2023).

[25] T. Hoefler, T. Haener, and M. Troyer, "Disentangling Hype from Practicality: On Realistically Achieving Quantum Advantage," *arXiv (Cornell University)*, Jul. 2023, doi: https://doi.org/10.48550/arxiv.2307.00523.

[26] M. Kraeling and L. Tania, "Software Development Process," *Software Engineering for Embedded Systems*, pp. 33–87, 2019, doi: https://doi.org/10.1016/b978-0-12-809448-8.00002-3.

[27] E. Bernstein and U. Vazirani, "Quantum Complexity Theory," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1411–1473, Oct. 1997, doi: https://doi.org/10.1137/s0097539796300921.

[28] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, Oct. 1997, doi: https://doi.org/10.1137/s0097539795293172.

[29] P. Kaye, An introduction to quantum computing. Oxford: Oxford University Press, 2010.

[30] C. Wohlin, M. Höst, and K. Henningsson, “Empirical Research Methods in Software Engineering,” *Empirical Methods and Studies in Software Engineering*, pp. 7–23, 2003, doi: https://doi.org/10.1007/978-3-540-45143-3_2.

[31] D. Jingle, S. Sam, M. Paul, A. Jude, and D. Selvaraj, “Design of Grover’s Algorithm over 2, 3 and 4-Qubit Systems in Quantum Programming Studio,” *International Journal of Electronics and Telecommunications*, Jul. 2023, doi: https://doi.org/10.24425/ijet.2022.139851.

[32] “Google Demonstrates Quantum Algorithm Promising Superfast Search,” *Popular Science*, Dec. 12, 2009. https://www.popsci.com/technology/article/2009-12/google-algorithm-uses-quantum-computing-sort-images-faster-ever/

[33] G. B. Arfken, H. J. Weber, and F. E. Harris, *Mathematical methods for physicists a comprehensive guide*. Amsterdam (Holanda) Elsevier, 2013.

[34] Shari Lawrence Pfleeger and J. M. Atlee, *Software engineering: theory and practice*. Upper Saddle River N.J.: Prentice Hall, 2010.

[35] A. M. Langer, *Guide to Software Development*. London: Springer London, 2012. doi: https://doi.org/10.1007/978-1-4471-2300-2.

[36] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, Introduction to algorithms. Cambridge, Mass.: Mit Press, 2009.

Биографија

Лука Радујевић је рођен 23. септембра 1998. године у Краљеву. У родном граду уписује основну школу „Браћа Вилотијевић“, коју завршава као носилац дипломе „Вук Караџић“. Школовање наставља похађајући природно-математички смер гимназије у Краљеву.

Након завршене средње школе, високо образовање започиње 2017. године на Факултету техничких наука у Новом Саду. Наредне године постаје носилац „Изузетне награде за успех“ Универзитета у Новом Саду. Током последње године основних академских студија постаје добитник стипендије „Доситеја“. Основне академске студије завршава у септембру 2021. године одбраном дипломског рада са насловом „Имплементација микросервиса за високоперформантно бројање учесталости придруживања кључева“.

Од 2020. године своју професионалну каријеру започиње као стипендиста компаније *TIAC* Д.О.О. у Новом Саду, где се специјализује у области развоја веб апликација.

Академску каријеру наставља 2021. године на Факултету техничких наука у Новом Саду похађајући мастерске академске студије на смеру „Инжењерство информационих система“.