Software Engineering for Blockchain Based Software Systems: Foundations, Survey, and Future Directions

Mahdi Fahmideh

University of Wollongong, Australia

John Grundy

Monash University, Australia

Aakash Ahmed

University of Adelaide, Australia

Jun Shen, Jun Yan, Davoud Mougouei, Peng Wang, Aditya Ghose, Anuradha Gunawardana

University of Wollongong, Australia

Uwe Aickelin

The University of Melbourne, Australia

Babak Abedin

Macquarie University, Australia

Many scientific and practical areas have shown increasing interest in reaping the benefits of blockchain technology to empower software systems. However, the unique characteristics and requirements associated with Blockchain Based Software (BBS) systems raise new challenges across the development lifecycle that entail an extensive improvement of conventional software engineering. This article presents a systematic literature review of the state-of-the-art in BBS engineering research from a software engineering perspective. We characterize BBS engineering from the theoretical foundations, processes, models, and roles and discuss a rich repertoire of key development activities, principles, challenges, and techniques. The focus and depth of this survey not only gives software engineering practitioners and researchers a consolidated body of knowledge about current BBS development but also underpins a starting point for further research in this field.

CCS CONCEPTS • **General and reference → Surveys and overviews • Software engineering, Software and its engineering, Software development process management, Blockchain**

Additional Keywords and Phrases: Software engineering, Systems development methods, Blockchain, Blockchain based software systems

1 INTRODUCTION

Blockchain-based software uses concepts and technologies popularized by cryptocurrencies such as Bitcoin – highly decentralized, open transaction ledgers with immutable content. Further advances, such as smart contracts and faster transaction processing, have provided advantages including transparent operation, user anonymity, auditability, and high scalability. A global survey of more than 1400 senior executives and practitioners across different countries by Deloitte [1] reveals that many IT-based organizations have found many compelling blockchain use cases in areas like finance, transport, healthcare, and manufacturing. A study by Chakraborty et al., [2] reports that over 3,000 blockchain software projects had been hosted on GitHub in March 2018 and that this number had been sharply doubled to nearly 6,800 in October of the same year.

However, in parallel with the growing interest in using and developing Blockchain-Based Software (BBS) systems, the accounts of project failure and system attacks in this domain have also become noticeable. For example, the MtGox attack in 2014 led to a declared loss of $600 million, Bitfinex attack in 2016 led to a loss of $65 million, and a DAO (decentralized autonomous organization) attack in 2016 caused the withdrawal of Ether digital currency funds worth $50-60 million. These failures are commonly attributed to a lack of adoption of a *systematic development approach*, a project management flaw that is aptly phrased as *un-ruled and hurried software development* by Porru and Pinna [3]. The situation has not significantly changed as blockchain projects in most organizations are in the experimental mode. A Gartner report in 2019 predicted that some blockchain projects would be replaced in the near future due to these development failures [4]. These negative outcomes of blockchain applications have been reflected in another Gartner

report [5] which mentions that *many chief information officers overestimate the capabilities and short-term benefits of blockchain [...] thus creating unrealistic expectations from blockchain platform vendors and service providers [....]*. Given these shreds of evidence, there is skepticism about whether blockchain technology really fortifies software systems. Research agendas, development frameworks, and prioritization of futuristic work have been put forward [6],[7].

To alleviate these issues, there is a need for more systematic software engineering approaches, sometimes deprecatingly referred to as engineering methodologies, to ensure the quality of both development and maintenance of BBS systems, which has been corroborated by prior research [3]. In this spirit, Chakraborty et al. [8] surveyed 156 active blockchain software developers from 145 blockchain projects hosted on GitHub and concluded that standard software engineering methods and best practices need to be adapted to address the unique characteristics of BBS and mitigate potential threats. In addition to characteristics such as decentralized architecture, unreliable connections, unexpected latency, and malicious system attacks that commonly occur in distributed computing systems including cloud computing [9] or IoT [10], BBS development raises unique challenges [11]. These challenges include a higher emphasis on security and reliability, accepting the immutability of system transactions compared to conventional data-intensive systems, and the need for an in-depth knowledge of cryptography, just to name a few [3]. Some researchers envision blockchain *as another class of software like the Internet* [12], and call for revisiting software engineering attuned with the requirements of this technology [2],[3].

In recent decades, the software engineering field has evolved to meet industry needs and to keep abreast of modern computing technologies that have influenced the development and adoption of software systems. In line with this, there is a growing awareness that the end-to-end development lifecycle of BBS requires new innovative guiding engineering approaches to bind together technical programming models, platforms (e.g., Ethereum), and technologies (e.g., Bitcoin scripting languages) [2],[3],[13]. Although small BBS projects may be manageable in an ad-hoc manner, a systematic engineering approach becomes essential if a BBS project is large and the BBS is aimed to support the core business processes of an organization. BBS components deployed on blockchain platforms are targets for malicious entities to attack. A systematic development approach can provide important activities, guidelines, checklists, design principles, and heuristics to mitigate, trace, and rectify the cause of vulnerabilities and errors. Hence, there is a need for a literature survey answering the looming questions like *how software engineering for the development of blockchain-based software systems is perceived according to the current state-of-the-art and industry efforts*.

Despite the growth and interest in the blockchain field, the software engineering aspect associated with developing this class of systems remains under-explored and under-conceptualized [3],[2]. Numerous studies are available in the literature including proposals, methods, empirical studies, and experience reports discussing different questions related to BBS development [14], [15]. The review, classification, and analysis of these different studies is a challenge for researchers and practitioners and demands rigorous and well-defined literature review methods. For this reason, we employed the search and selection protocols proposed by the procedure of *Systematic Literature Review* (SLR) [16] and qualitative data analysis based on *Grounded Theory* [17] to provide a complete review of the state-of-the-art of BBS development. Our literature review considers 58 selected core studies (Appendices A and B) which are organized into a new conceptual framework. The framework addresses different aspects of BBS development, in particular, *approaches*, *processes*, *modeling*, and *role* which guide software developers, business managers, and academic researchers in the exploration of practical side, implications, and challenges related to BBS development. To the best of our knowledge, our survey is an unprecedented work and a novel attempt at summarizing the state-of-the-art research and practice on end-to-end engineering lifecycle for BBS. As discussed later, no other published surveys match the scope, depth, and coverage of our survey and its findings. The key contributions of this work are:

—a coherent set of method fragments grounded in the literature to facilitate a better understanding of the BBS engineering lifecycle;

—a guidance framework, as *having a tail-light to follow*, for novice software practitioners engaging in BBS development context; and

—an outlook on open challenges in BBS development for future research directions.

The rest of this paper is organized as follows. In Section 2, the discussion of blockchain concepts lays the basis to raise essential implications for the development of BBS. Section 3 summarizes past literature work related to our current survey. In Section 4, we introduce our proposed conceptual framework, which is derived from the extant literature on BBS engineering and provides a lens for an analytical literature review. Section 5 delineates our findings

based on our framework, followed by a discussion on existing knowledge gaps in the literature in Section 6. We close with a discussion of the limitations of this survey in Section 7 and a concluding remark in Section 8.

2 BACKGROUND

This section introduces the core concepts including the building blocks for BBS (Section 2.1) and the needs for software engineering for BBS (Section 2.2). The concepts and terminologies introduced in this section are used throughout the paper to guide the discussion of the survey results.

2.1 Building Blocks for Blockchain

New system engineering approaches are typically grounded on fundamental concepts and underlying logic. As depicted in Figure 1, blockchain is commonly rooted in a few core elements that a BBS is built upon [12], [13], [18], [19]. A *distributed ledger* is a form of a shared database that may exist across multiple locations and among several participants [13]. It enables parties to authenticate, process, and validate transactions without the need for central authority or intermediary. *Blockchain*, with its origin in the Bitcoin cryptocurrency [18], is a type of distributed ledger. It is viewed as a promising option for the secured and reliable next generation of Internet-Based information technologies [19]. Some industrial domains have already developed BBS and others are still figuring out a reasonable use case to offer new BBS for transactional digital services [13]. In a simple view, blockchain is an accounting book or digital distributed database. It has a chain of *blocks* (i.e. records) that are sequentially linked together. Each block depends on its predecessor block and is secured via cryptography techniques [19]. A block contains transactional data, a time stamp, and a hash value of its previous block. Also, a block saves an arbitrary set of transactional data that is created by a *node* (a computer in the distributed ledger network). The chain of blocks is stored on a distributed network of nodes where each node contains a copy of the entire blockchain. The chain is visible and verifiable by all nodes participating in the network. Once a block with its own time-stamp is appended to the chain, the creator node broadcasts that block to all the other nodes in the peer-to-peer distributed network. Once nodes receive the block, they validate it via predefined check, and add the block to their own local blockchain copy to provide a single source of truth. Some nodes, called *validators*, are responsible for validating a newly added block to the blockchain. The data records in a block are non-reversible, transparent, and become an immutable part of blockchain after they are accepted by all nodes. Such a chain of blocks provides a secured means of information exchange between systems without involving a trusted third-party and is suitable for record-keeping tasks such as financial transactions, medical records, and so on [13]. Casino et al. [20] predict that blockchain technology will continue to be adopted by multiple industries such as manufacturing, cybersecurity, 5G networks, and IoT to augment security, track records, and ensure data consistency.

A key element of blockchain technology is the ability to create *smart contracts*. The term, which was coined by Nick Szabo in the mid-1990s [21], is defined as translating the clauses of a business contract into code and embedding them into software or hardware to make them self-execute. This reduces the cost of contracting between transacting parties and avoids malicious actions during contract execution. Different alternative definitions of smart contracts have been already proposed. For example, Crosby and Pattanayak [18], view a smart contract as database slots that reserve the necessary logic for the creation and validation of transactions and enable users to read, update, and delete the data stored in blockchain platforms. A broad definition of smart contract that covers the breadth of these definitions is proposed by Clack et al. [22]. That is a smart contract as *an automatable and enforceable agreement. Automatable by computer, although some parts may require human input and control. Enforceable either by legal enforcement of rights and obligations or via tamper-proof execution of computer code* [22]. Smart contracts are either implemented via domain-specific languages, e.g. Solidity on Ethereum platform, Liquidity on the Tezos platform or by general-purpose languages, e.g. Java, Go, Kotlin to ease contract programming. For instance, Ethereum platform offers the built-in language Solidity and executes the compiled bytecode of Solidity scripts on its execution environment, called Ethereum Virtual Machine (EVM). A smart contract's script code is invoked if it receives a transaction from a user. The smart contract defines necessary entry points of transaction execution. In the terminology of Ethereum, each entry point is called a *function*. A transaction specifies entry points at which function codes of a smart contract should be executed. In line with this, a BBS is developed by codifying business logic into smart contracts. The smart contracts are then deployed and autonomously executed on decentralized ledgers, e.g.

Ethereum. This end-to-end development needs a systematic development lifecycle or software engineering methodology [3].
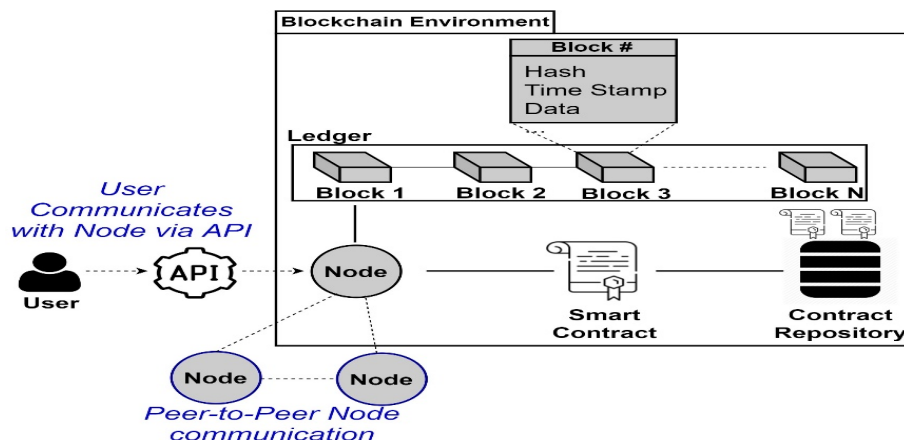

Fig. 1. Fundamental constituents of the blockchain architecture

2.2 Software Engineering for BBS system

We need to first establish an understanding of the meaning of software engineering for BBS systems. A simple definition, according to Porru et al., [3], is that BBS *is a new type of software that uses the implementation of blockchain in its components*. If a system depends on blockchain, its development may *revolve around this technology* [3] and be perceived as *actually different from a non-BBS development* [2]. Unlike non-blockchain software systems, the development of BBS systems raises new challenges and difficulties that software engineers have to cope with. As explained in Section 5, these challenges highlight the importance of further research in this field in both technical and non-technical elements, such as better guidance for blockchain developers, a trade-off between security and performance, the choice of consensus protocols in relation to transaction time and computing power requirements, the cooperation of multiple institutions and stakeholders, [2],[7],[19]. Some of these issues can be attributed to the immaturity of the technology itself as it is common in all new technologies at their inception. Others are intrinsic to blockchain concepts and requirements. Conventional wisdom in software engineering acclaims that the appropriate use of engineering methodologies ensures the development of quality software that meets expected outcomes while satisfying budget and timeframe constraints [23]. Inspired by the lessons learned in software engineering industry informing that *the more a system gets complex, the more it is open to total breakdown* [24], it can be suggested that understanding and adopting systematic engineering approaches for BBS reduces the likelihood of user dissatisfaction and unforeseen security and other errors that may become costly to fix. Drawing on the common prudence in software engineering field [25],[26],[27], without a systematic approach it will be hard for a software team to dissect and trace the cause of errors and correct them when things go wrong. In other words, the quality of development significantly impacts the quality of the resulting BBS running on blockchain platforms [2],[3].

3   RELATED SURVEYS

Due to its popularity, many literature reviews on different aspects of the blockchain topic are available. In Table 1, we broadly classify the related studies into *broadly-related surveys* focusing on software engineering for Internet-based distributed computing applications and *closely-related surveys* pertinent to blockchain applications.

3.1   Surveys on Engineering Internet-Based Computing Systems

Broadly-related surveys focus on intersections and synergies between Internet-based computing technologies such as blockchain, service computing, cloud computing, and IoTs. They are theoretically grounded on distributed architecture engineering that provides a backbone and virtually unlimited computational resources and on-demand services to develop and deploy software systems. Surveys of Fahmideh et al., [9], [10], Lane et al., [28], and Razavian and Lago [29] are categorized under the broadly-related surveys. For example, cloud computing empowers the

scalability and performance of blockchain platforms by providing service delivery models such as Infrastructure as a Service (IaaS), Software as a Service (SaaS), or Platform as a Service (PaaS). Fahmideh et al. in their initial work [30] that is later elaborated in [9] compare the characteristics of 43 reengineering approaches, published between 2007 and 2015, to make legacy software systems as cloud-enabled. Their survey synopsizes the status quo and presents a rich set of development tasks and recommendations commonly incorporated into typical migration processes to cloud platforms upon service delivery models such as IaaS, SaaS, or PaaS. Blockchain is used as an enabling technology for IoT-based applications as it provides a secured decentralized network for data management and communication of sensors and devices. Fahmideh and Zowghi [10] present a generic development process lifecycle for IoT-based applications by mapping 63 approaches, published between 2008 and 2019. They conclude that development roles, requirements analysis, modelling, testing, and tailorability of development processes are not addressed in the existing approaches to engineer IoT-based applications.

Razavian and Lago [29] present an analysis of reengineering processes in 75 approaches for developing SOA (service-oriented architecture) based software applications. Their main contribution is to devise a new conceptual model and classifications of activities such as code analysis, architecture recovery, service design, and implementation in order to integrate with and componentize legacy applications to web services. They identify major challenges of (i) value creation for service-based software engineering, (ii) decision making on tool selection, and (iii) legacy understanding without reverse-engineering.

In addition, a possible interplay between blockchain and big data is to use blockchain for storing big data analytics software applications that enable user authentication, recording data access history, and proper use of encrypted data on distributed peer-to-peer networks. Davoudian and Liu [31] review challenges related to three major software engineering activities of requirements analysis, design and implementation in the context of big data analytics development based on academic and multi-vocal literature. Despite the focus on software engineering of systems relying on Internet-based computing technologies, all the above surveys do not directly deal with technical implications and complexity of software engineering aspects highlighted in BBS such as adopted approaches, processes, models, and roles that represent the multifaceted view of development lifecycle as presented in our framework in this survey. Despite their usefulness in forming a basis for our conceptual framework, we deem that the recent literature surveys on conventional software development processes and improvement, e.g.[32],[33], do not overlap with the focus and scope of our research objectives.

3.2 Surveys on Blockchain-based Systems Engineering

Scant work is available as being exclusively devoted to a review of existing proposals on the engineering of BBS. To support in-depth technical analysis, we discard introductory surveys that aimed at demystifying the notion of the blockchain and discussing prevailing challenges in adopting this technology in software applications and organizations. The reason is that they do not concentrate on the aspect of the development lifecycle and despite their usefulness in crafting our research objective, they fall outside the scope and focus of this survey. Some example surveys in this genre are the taxonomy-based surveys on the blockchain usage trend in the IoT context [34], consensus algorithms and application domains of blockchain [19], security concerns in blockchain adoption [35], [36], and business applications of blockchain [37]. The most extensive work is provided by Yli-Huumo et al., [14] with a research objective to *understand the current research topics, challenges, and future directions regarding blockchain technology from the technical perspective*, (page.1). By analyzing 41 primary studies, published between 2012 and 2015, where 80% of the papers focus on Bitcoin systems and less than 20% deal with other application domains, Yli-Huumo et al. highlight that the lack of concrete evaluation metrics and techniques to measure scalability efficiency in terms of throughput and latency is left unaddressed in the literature. The goal of the survey by Liu et al., [15] is to investigate the design of smart contracts, implementing trusted transactions among parties without mediators. They selected 53 papers to show the state-of-the-art of this topic, developed a taxonomy towards the security verification of blockchain smart contracts, and discussed the pros and cons of each category of the related studies. Vacca et al. [38] review 96 articles published between 2016 and 2020 to identify existing solutions in tackling software engineering-specific challenges related to the development, test, and security assessment of blockchain-oriented software.

None of the surveys in Table 1 is specifically geared towards the development lifecycle for BBS and intrinsic characteristics of blockchain that challenge well-established development processes of conventional software

systems. As a response, we have narrowed our focus to extant studies that propose completely or partially, end-to-end approaches for BBS development. This includes development themes, essential tasks for incorporation into the development process, emerging roles, and modeling challenges. From this angle, our survey is more precise when compared to the related surveys. Our literature analysis is performed through a conceptual framework, encompassing four important dimensions of development approaches, processes, modeling, and roles that are not covered in related surveys. Our study aims to aid software teams, managers, and researchers in characterizing the extant material in the literature and mastering skills necessary to develop BBS. Due to the different focus and scope of our review, many research papers that we discuss in this survey are not covered by these existing surveys. For instance, none of the studies reviewed in our survey is covered by [15]. Our work supersedes the existing surveys as it considers many recently published studies in blockchain literature that are not included in the related surveys.

Table 1. Comparison of proposed survey with existing related surveys

	Survey reference	Computing paradigm	Research focus	Publication year	Reviewed studies	Study type
Widely-related surveys	Razavian and Lago [29]	SOA	Reengineering processes of legacy software systems to SOA	2015	75	Systematic literature review
	Fahmideh et al. [9]	Cloud	Reengineering processes of legacy software systems to cloud platforms	2016	43	Systematic literature review
	Davoudian and Liu [31]	Big data analytics	Challenges associated with software engineering activities of requirements analysis, design and implementation of Big data analytics software applications	2020	Not stated	Literature survey
	Fahmideh and Zowghi [10]	IoT	Development processes for IoT based systems and platforms	2019	63	Systematic literature review
closely-related surveys	Panarello et al. [34]	IoT and blockchain	Integration of BC and IoT applications in terms of different application domains, usage patterns, device manipulation, and data management	2018	Not stated	Literature survey
	Zheng et al. [19]	Blockchain	Blockchain architecture, key characteristics of the Blockchain, consensus algorithms and protocols	2018	Not stated	Literature survey
	Mohanta et al. [36]	Blockchain	A comprehensive analysis on blockchain applications including implementation challenges associated security and privacy	2019	135	Literature survey
	Li et al. [35]	Blockchain	Real security attacks to popular blockchain systems	2020	Not stated	Literature survey
	Konstantinidis et.al [37]	Blockchain	Business applications of Blockchain in both public and private sectors	2018	44	Systematic literature review
	Liu et al. [15]	Blockchain	Security assurance and correctness verification of smart contracts	2019	54	Literature survey
	Vacca et al. [38]	Blockchain	Challenges in implementing, testing, and secure-aware smart contracts	2021	96	Systematic literature review
	Our survey	Blockchain	Blockchain software engineering	-	59	Systematic literature review

4 RESEARCH APPROACH

Due to reasons of (i) the sheer volume of published material in an overgrown and unkempt domain like blockchain and (ii) the exploratory nature of this literature review, we employed the kernel techniques from software engineering and information system research fields. In particular, we used the guidelines for conducting a *Systematic Literature Review (SLR)* [16] and qualitative data analysis based on *Grounded Theory* [17] as outlined in Figure 2 and

detailed in sections 4.1 and 4.2. We want to provide a conceptual framework, encompassing commonly occurring fragments of software engineering methods for BBS and their interrelations. In the following, we describe our literature review procedure and the steps to derive the conceptual framework, guided by the illustration in Figure 2.
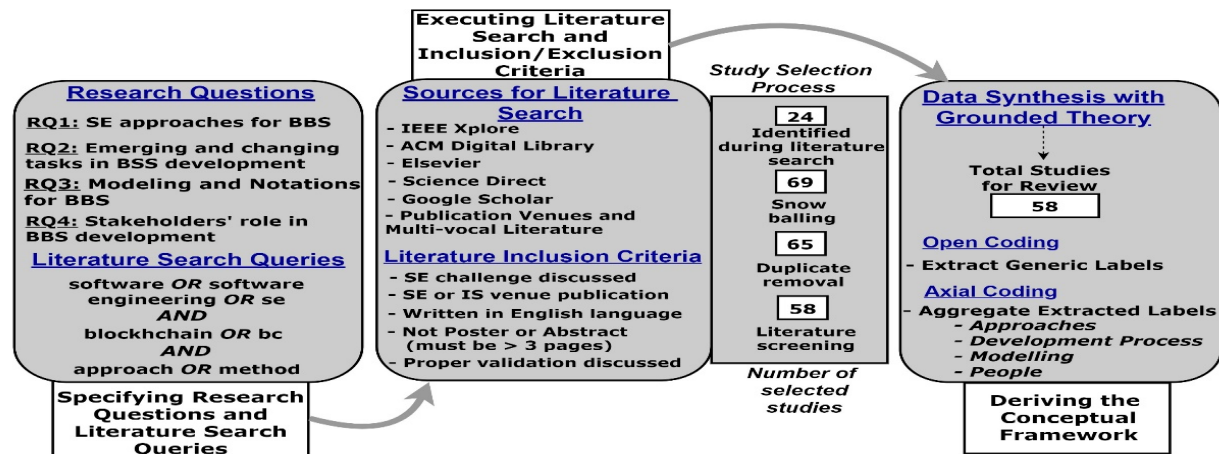

Fig. 2. Sequence of steps undertaken in our literature survey

4.1 Systematic Literature Review

As stated earlier, this study aims to portray the current research landscape in software engineering for BBS. We formulated the following research questions:

- *RQ1: what are the common software engineering approaches that are adopted for BBS development?*
- *RQ2: what are the key emerging and changing tasks in developing BBS systems?*
- *RQ3: what software modeling approaches and notations are applicable in BBS development lifecycle?*
- *RQ4: what key roles are in a BBS development endeavor and what do they play?*

We followed the well-known and accepted procedure and guidelines by Kitchenham et. al. [16] to conduct this SLR as they deem to be an appropriate way to systematically identify and qualitatively specify the criteria for the selected studies involved in the review and data synthesis. In the blockchain field, the concepts, nomenclature, and terminologies have not been consolidated yet. Hence, our SLR undertaking was not a linear and mechanical procedure. Instead, our review tended to be hermeneutic and iterative. That is, to get immersed in the literature, as recommended by Brun et al. [39], we read introductory and literature survey papers, some of which are listed in Section 3 (Table 1), to get an in-depth understanding of the important aspects of BBS engineering and further refinements of the literature review, in particular, inclusion/exclusion criteria definition and data extraction step. Our SLR's key steps included review protocol, searching queries, selection of study sources, study selection, data extraction, and data analysis as shown in Figure 2. More exactly, we determined the main terms *software engineering*, *blockchain*, and *approach* based on the research questions to define a set of search queries along with related abbreviations, plurals, and extended them with alternative synonyms. The *approach* was also used because some papers from our initial literature search used these terms to express frameworks and methods for the development of BBS. Search queries were executed against the main scientific digital libraries including Google Scholar, IEEE Explore, ACM Digital Library, Elsevier, SpringerLink, and ScienceDirect. We then selected papers that met the following inclusion criteria:

- explicitly discussed software engineering challenges related to BBS;
- published in the software engineering and information systems international venues such as journals, conferences, congresses, and workshops as well as computing specific venues, e.g. computer networks and sensor-driven systems, that streamline software aspects of BBS;
- written in English language with at least three pages (not an extended abstract, and
- provided a proper validation of their work such as case study application, exemplar scenario, interview/survey of practitioners of blockchain practitioners, simulation, comparative analysis, or theoretical evaluation.

Thus, based on the criterion above and standard SLR practice we discarded studies that were

- opinions, white, and short papers without any sorts of evaluation results;
- provide only background on blockchain or discussed the applications of blockchian in other enabling technologies like IoT, big data, and cloud computing;
- Thesis or other non-refereed publications; and
- Not written in English

In addition to the digital libraries, we paid attention to peer-reviewed venues exclusively dedicated to blockchain publications themes such as *International Conference on Blockchain* and *International Conference on blockchain and Trustworthy Systems*. We also sought online non-academic literature, i.e. multi-vocal literature [40], such as internet blogs, white papers, and trade journal articles, which could share real-world findings surrounding BBS development. A total of 24 papers were identified from the previous search iterations in the selected digital libraries. An important technique that we strictly used during the literature search iterations was *snowballing* [40], through which we identified 65 new important papers from the reference sections of the initially identified papers. In the last search iteration, snowballing technique resulted in 69 papers that were reduced to 65 after removing 4 duplicated papers from the same authors. We screened the title, abstract, and preferably full texts of the papers, and shortlisted 60 papers after applying the inclusion criteria. To avoid being bias, the list of the identified papers by the main research investigator was double-checked by two co-authors, independently. Two papers were removed after they were found as unsatisfactory to the inclusion criteria and the objective of this survey. The literature search iteration in its last round culminated in 58 papers listed in Appendix A. A summary of the identified papers' demographic information (e.g., types and years of publications, types of validations, and geo-distribution of authors) is shown in Appendix B. We denoted each paper (S for short) by combining a number to distinctly refer to it (i.e., [Sn], n is unique a number from 1-58) throughout this survey and conceptual graphs in Section 5.

4.2 Conceptual Framework for Software Engineering of BBS

In light of different terminologies and broad varieties of blockchain concepts, it is essential to first establish a foundation to enable our analysis of literature pertinent to software engineering for BBS. To this end, we devised a conceptual framework that, according to Miles and Huberman [41], would enable researchers to graphically explain the main concepts and relationships in a domain of interest. In this survey, the derivation of such a framework was based on Grounded Theory [17], which has been found useful in qualitative research and its application is well-recognized in several fields including medical sociology, education, and management. In recent years, using Grounded Theory in the software engineering field has received growing attention [42]. It aims to generate theories and conceptual models out of a collection of data about a research problem and to enable researchers to have amenable interpretation and free comprehension of data in different ways. The data can be sourced from, for instance, field observations, interviews, secondary data, semi-structured data, pictures, and diagrams. The generation of the theory from the data is based on a few steps including *open coding*, *axial/selective coding*, and *theoretical coding* that can be, according to Strauss et al. [43], governed and consulted by the pre-defined concepts/views from a domain literature to enhance theoretical sensitivity. We took into account the literature on conventional software engineering [44] to analyze the identified papers. We used open coding to break down and label data into smaller fragments [43] to analyze and highlight the text segments in each paper that could be considered as a discrete piece of data annotated with a descriptive *label*, as a series of high-level *intellectual bin* [41]. The labels could help structure and focus on the context of the data in the papers. As an example to demonstrate the inner working of open coding, Table 2 presents an excerpt of text segments that were extracted from papers [S24], [S25], [S26]. When these quotes are collectively viewed, their common theme, despite the difference in wording, is related to the notion of the suitability of blockchain, which should be performed by software teams at the early stages of the development process. Thus, we defined a label in the conceptual graph of the framework as a development task and named it *feasibility analysis of BBS* supported by the identified papers, which will be delineated in Section 5.2.1.

Table 2. An example of emerging development task named *feasibility analysis of BBS* via open-coding

Quote from the source paper	Source
This technology is being applied to an increasing range of industries and problem spaces. However, such an application might not always be appropriate or optimal; in many cases [...] while evaluating potential applications of blockchain at CableLabs, [...] I developed a framework for determining whether blockchain is appropriate [...]	[S24]

Table 2. An example of emerging development task named *feasibility analysis of BBS* via open-coding

Quote from the source paper	Source
Investigating the feasibility of developing a smart contract that manages a network of banks [...] is a big challenge, above all looking at the throughput of the blockchain and hence to the rate at which a blockchain can confirm transactions.	[S25]
Due to the long commit time and high transaction fees on a public blockchain (where fees are largely independent of the transacted amount), it is often infeasible to store every micro-payment transaction on the blockchain network. On-chain transactions are suitable for transactions with medium to large monetary value, relative to the transaction fee.	[S26]

The open coding step resulted in labels such as *requirements analysis for BBS*, *state management design*, *replication and synchronization design*, *authentication and authorization design*, *interaction design*, *smart contract design*, *consensus mechanism design*, and *incentive mechanism design* each of which is supported by at least one reference to the identified source. Once labels emerged over 58 identified papers as the source of data, axial coding was performed to aggregate and condense related labels to form broader groups of labels. We derived four essential groups of labels, a.k.a. aspects, namely (i) *approaches*, i.e. kernel theories or software engineering foundations that could govern the end-to-end BBS development lifecycle, (ii) *development process*, (iii) *modelling*, and (v) *roles*. More specifically, some labels were grouped as a kind of *modelling* aspect: a *smart contract model*, recommended by studies [S1], [S5], [S31], [S46], [S56], and [S57], that is used to represent the expected functionalities and features of smart contracts implemented and tested by low-level code scripts running on blockchain platforms. Axial coding resulted in labels related to the aspect of *role* that is involved during a BBS development endeavor. If a label couldn't be classified under any or indirectly related to one or more of these aspects, we could define a new label or group it under a related aspect. That is, some labels were related to decision factors for incorporation into the analysis and design activities, e.g. *decision on-blockchain and off-blockchain components*, *decision on blockchain type*, and concerns e.g., *visibility and transparency*, *organization restructuring*, *uncertain status*. Axial coding was further elaborated, in line with the literature on conventional software engineering [44], to create new high-level labels that could classify fine granular labels with common themes under the same aspect. For example, tasks *feasibility analysis of BBS* and *requirements analysis for BBS* were subsumed under the *BBS analysis phase* in an analogy with the analysis phase in conventional software engineering. Often, there should be references supporting labels in this conceptual framework generated from Grounded Theory. These references to the identified papers are listed in the conceptual framework's graphs and cited in Section 5 wherever we present the results of the literature review. It should be noted that a paper could belong to multiple aspects of the framework. Grounded Theory was ended with theoretical coding where we established the conceptual relations between substantive labels, resulting in the conceptual framework for BBS engineering. Using the Grounded Theory in this literature survey was not intended to develop testable hypotheses nor to answer unexplored research queries in the blockchain literature. Rather, we used it to (i) analyze and compare the extant papers in the literature on BBS engineering, (ii) identify existing engineering challenges and corresponding techniques, and (iii) construct further research directions for unresolved challenges. Our generic conceptual framework, yet pluralistic and platform-independent, binds the essential aspects together, provides an integrated overarching view of software engineering for BBS to facilitate understanding of existing challenges and solutions proposed by blockchain community in the literature, and enables further research and development in this field. The next section presents the framework that was produced to structure a coherent set of relevant aspects of BBS engineering.

5   RESULTS

To present our results in an illustrative manner, we use visual conceptual framework diagrams as represented in Figures 3,4,6,7, and 8. In this conceptual framework, a *node*, either sub-node or parent node, may refer to a task/sub-task, technique, decision factor, challenge, modeling, or role that comes into play. *Directed edges* (or a*rrows)* show relationships among nodes. The nodes and arrows are supported by one or more source studies where the full demographic information of the study set is listed in appendices A and B. As mentioned in Section 4, rooted in the fundamental aspects of software engineering, the diagrams in the framework characterize BBS development based on the four major aspects of *approach*, *process*, *modeling*, and *roles*, which are, respectively, related to answering RQ1, RQ2, RQ3, and RQ4 (Section 4.1). These allow software teams to explore and make a comparison between different BBS development options. These aspects and their importance are substantiated below.

The aspect of *development approach*, i.e., theoretical foundation, is rooted in the fact that software engineering endeavors are not impartial on their own, rather they have underlying development perspectives, including principles, assumptions, and drivers that govern and rationalize an end-to-end development lifecycle in a given project context [45]. BBS development is thus founded on some fundamental concepts and basic logic such as trust, decentralized and distributed public/private ledger, cryptography, smart contract, consensus algorithm, and transaction transparency that should take into account by software teams. These are critical and deserve close investigation.

The aspect of *process* specifies guidance for development lifecycle phases, necessary tasks, and techniques that are sequenced into the development of BBS. We borrowed the generic phases of the software development lifecycle (SDLC), i.e. *analysis*, *design*, *implementation and test*, and *maintenance*, introduced by Pressman [15] allowing us to organize and represent the software engineering process body of knowledge. The rationale to incept BBS development process like SDLC is that, regardless of an application domain or adopted underlying blockchain platforms, it is generally taken as axiomatic that engineering processes are almost the same at the macro and abstraction level, though they differ in fine-granular development tasks and technical-centric implementation details. This interpretation is consistent with previous exploratory literature surveys in other Internet-based computing domains such as IoT [10], cloud migration processes [46], and SOA based development processes [47],[48] where the authors leverage SDLC phases to classify peer-reviewed studies to compare and contrast them and to identify unaddressed research gaps. This aspect enables a better understanding of BBS life cycle management and it is populated with the identified commonly occurred development process tasks from the literature.

The aspect of *modelling* deals with the representation techniques and notations that developers may employ for process tasks' outputs during BBS development endeavors. This aspect, itself, encompasses two sub-aspects of (i) a *model*, a.k.a. *work-product/artefact*, as a result of development tasks according to Gonzalez-Perez et al. [49], and (ii) a *modelling language* to represent and maintain generated models. A model, as generally defined by ISO/IEC 24744 standard meta-model for software engineering methodologies [50], is an artefact of interest to a software team. In a blockchain project context, this may be a legal document, distributed ledger architecture, Bitcoin transactions, smart-contract template, Solidity code, or a piece of information related to the project. On the other hand, a modelling language provides underlying elements to represent a model. For example, class, inheritance, aggregation, or composition in the object-oriented software development paradigm enables developers to express different structural and behavioral facets of a model at a different level of abstraction serving as overlapping aspects of system development.

The *role* aspect indicates associated responsibilities, interactions, and required skills in a BBS development endeavor. This aspect in our framework is required as the development of BBS relies on different technical expertise, business vision, and delivery skills that imply having a development role acquisition plan. Bosu et al. [S22] suggest that special skills, beyond those known ones in non-BBS development, are needed for people considering joining BBS projects. There are advantages in defining the aspect of roles participating in BBS development. That is, specifying BBS roles and their duties enables the separation of concerns so that if changes occur to one role, there will be less influence or dependency between roles. Moreover, this clarifies the responsibilities of each stakeholder, enables better team management, and monitors activities per role. Another advantage of defining the role aspect in our framework is to inform software teams who are with limited experience in blockchain and are not quite sure about the roles required in BBS development.

5.1 RQ1: What are the common software engineering approaches that are adopted for BBS development?

In the identification of proposed approaches for BBS development in the literature, we did not set any pre-defined classification during the open coding step of Grounded Theory. Instead, we found approaches based on what appeared in the selected studies. We observed five distinct themes, as depicted in Figure 3, that BBS development may leverage, namely (i) *Agile based development* (e.g. [S1],[S4],[S16],[S23],[S27],[S28],[S51])*, (ii) *model-driven development (MDD)* (e.g. [S5],[S16],[S28],[S57],[S29],[S30],[S34],[S52]),(iii)*architecture-based BBS development* (e.g. [S3],[S4],[S11],[S12],[S33])*, (iv) *pattern-based BBS development,* and (v) *ontology-based BBS development*. The adoption of these approaches is not mutually exclusive, rather they can be used together and orthogonally used across the development lifecycle. For example, in [S16], authors have shown the model-driven and Agile-based approach for BBS.
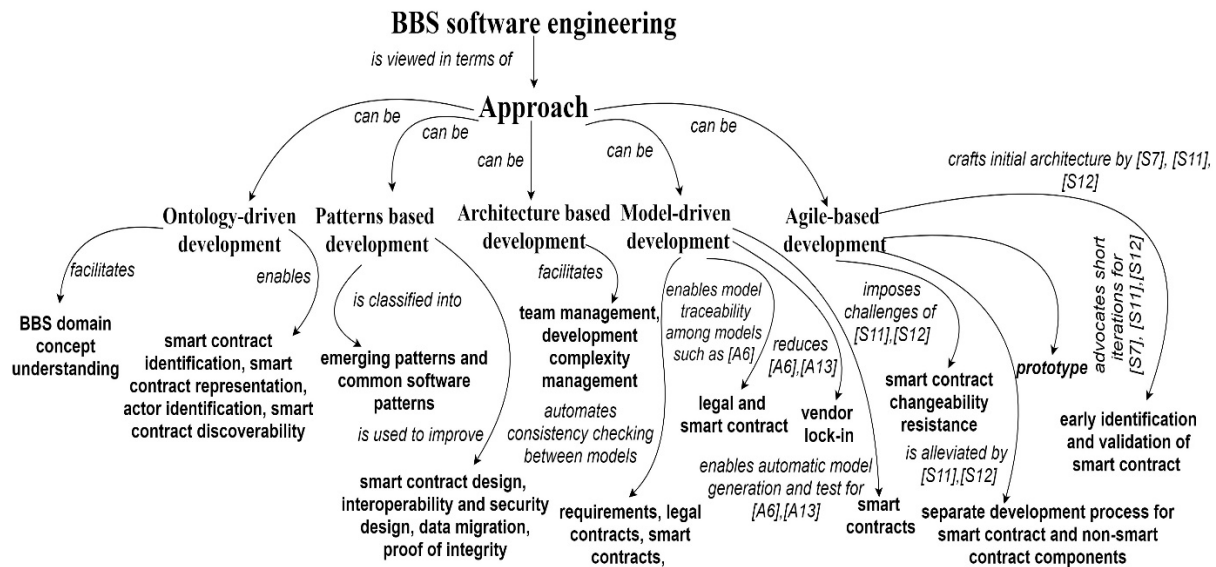
Fig. 3. *Approach* conceptual map

5.1.1 *Agile-based BBS development*

The common perception of agile-based software development is distilled in the *Agile Manifesto* [51] and used in well-known agile software development methodologies such as SCRUM, XP, and Lean Kanban. They emphasize practices such as active user involvement, lightweight modeling, short cycles, iterative releases with frequent customer review, and priority of working software over documentation. Blockchain researchers have utilized agile practices for non-real-time critical BBS development [S1],[S4],[S16],[S23],[S27],[S28],[S51]. BBS Agile prototyping, a.k.a. Spike Solution in XP, has been found to be a useful technique at the early stage of BBS development for the purpose of (i) requirements elicitation, specification, and thus far from target blockchain platform, (ii) forming a base target BBS architecture without dependency to a specific platform, and (iii) identifying uncertainties in system quality factors such as transaction execution performance, security, as well as the trade-off between them [S1],[S4],[S28]. Marchesi et al., [S23] discuss the usefulness of *user stories*, i.e. defining target BBS features as seen from stakeholders' point of view, to get early feedback from stakeholders to validate required smart contracts. Being open to users' change requests and continuous integration, as promised by core Agile development technologies like DevOps [52], nevertheless, is not consistent with the intrinsic characteristics of blockchain harness. For example, the data that is generated and stored by smart contracts on distributed ledgers is almost impractical or too costly to modify by external users and developers. This limitation is mainly due to the immutable and append-only feature of blockchain environments in which smart contracts are executed [12],[13],[18],[19]. As will be discussed in Section 5.2.1, a conclusion from this matter is that Agile practices are thus useful at the early stage of BBS engineering to identify potential BBS requirements and smart contracts. They may not be always suitable for developing ever-changing business services. Rather, software teams should implement smart contracts for business services with minimum later upgrade needs at run-time and ensure the verification of smart contracts before their deployments on blockchain platforms. In this regard, [S23] suggests separating the system development lifecycle into two individual concurrent development endeavors, i.e., (i) smart contracts development and (ii) typical software development. Complementing this, developers should implement proxy/mediator components to enable interactions between (i) and (ii) and to inform on-blockchain components of changes in off-blockchain components.

The benefits of agile development and blockchain are reciprocal. Whilst the above-mentioned studies narrow their focus on the contributions of agile practices for the engineering of BBS, other studies explore the ways blockchain features enhance and scale to the agile development. For example, blockchain can support the communication between product owner and developers in handling product acceptance test and certification via smart contracts in agile Scrum or Lean-Kanban process as discussed by Lenarduzzi et al., [53]. The benefit of blockchain to agile development falls out of the scope of this survey, although, it is an important potential research area.

5.1.2 Model-driven BBS development

The underlying principle of model-driven development (MDD) in software engineering is *abstraction* through which separation of concerns can be maintained between abstraction (domain models) and specification (implementation models) along with necessary transformation rules to move from abstraction to specification [54]. The abstraction provided by models facilitates understanding of a given domain and automatic code generation while maintaining the traceability between models (system design) and code (system execution) [55]. A model can express the structure and behavior of domain concepts and define a set of implying statements about concepts and their relationships [56]. Numerous advantages have been shown for incorporating MDD into BBS engineering [S5],[S16],[S28],[S57],[S29],[S30],[S34],[S52]. Firstly, MDD-based development platforms, e.g. Ethereum and Hyperledger Fabric (See section 5.2.3), offer model transformation techniques and generate well-tested smart contract bytecodes to automatically detect vulnerabilities that may be exploited by attackers targeting blockchain platforms. Secondly, platform-agnostic models of smart contracts mitigate the issue of vendor lock-in to a specific blockchain platform as each platform may have its own programming language to implement smart contracts, which reduces the reusability and platform interoperability of smart contracts. If a developer needs to deploy the current smart contract, for example, written in Solidity language of Ethereum platform in Neo platform based on C# language, he/she needs to learn a new language and re-implement smart contracts. By following a MDD-based approach, once a smart contract is modeled, it can be transformed to multiple blockchain platforms. Thirdly, the traceability to requirements and consistency checking, for instance, from legal contract models to smart contract models and subsequently to smart contract code models is facilitated. Fourthly, from a human perspective, smart contract models are easier to understand than smart contract codes. Hence, they improve communication in/between blockchain engineers, domain experts, and stakeholders.

The abovementioned benefits can be embodied throughout a BBS development endeavor. An example of this application is the work by Jurgelaitis et al. [S5] that proposes an end-to-end development lifecycle. This work is inspired by the Model-Driven-Architecture (MDA) [57] approach, which defines three types of models that are gradually refined from high-level models to platform-specific ones based on vertical model transformation techniques. The authors use Computation Independent Models (CIMs) to represent BBS requirements and interaction models. CIMs are used to derive Platform Independent Models (PIMs) representing, in particular, high-level smart contracts that are bounded to implementation details, e.g. Ethereum or Hyperledger. PIMs, give a broad view to software teams of what smart contracts should be in operation, help with a better selection of target blockchain platforms, specify transaction structures, participants, and consensus algorithms which are next reflected in Platform-Specific Models (PSMs). Model transformation rules are used to generate smart contracts for a chosen platform out of PSM models. In adopting MDA harness, software teams should identify what needs to be modelled and how it to be transformed. The discussion on the challenges of model transformation for the automatic generation of smart contracts is an important potential research area as discussed in the selected studies [S5],[S16],[S28],[S57],[S29],[S30],[S34],[S52], however, it falls out of the scope of this survey.

5.1.3 Architecture-based BBS development

In software engineering, architectural thinking is a means for managing software development complexity by breaking down a large system into its components, their interrelationships, existence rationale, and important properties. Software architecture models, such as *4+1 view* [58], enable the representation of different software architecture aspects such as logical, process, physical, and scenario. This enables examining if system quality factors are achievable [59]. The use of an architecture-centric approach in BBS development, as reported in [S3],[S4],[S11],[S12],[S33] is recommended for the purpose of complexity management. For instance, a logical view of the target BBS architecture, suggests decomposing BBS into different *layers* for separation of concerns where each layer is allocated to an individual team. The architectural view by Glaser et al. [S33] divides BBS development into three layers: (i) *presentation layer* where end-users interact with the BBS, (ii) *fabric layer* that fulfills basic services and the actual blockchain code base including communication layer, the public key infrastructure, data structures, and constructs for developing and execution of smart contract languages, and (iii) *application layer* that includes software components invoking basic services from the fabric layer. In a similar vein, [S4] and [S11] jointly define layers namely *application*, *trust*, *blockchain*, *transaction*, and *network*. Alternatively, developers can divide BBS architecture into *modules* to be designed and developed like (i) *REST API/front-end*, (ii) *blockchain Hyper-ledger*, and

(iii) *back-end system*, where REST API/front-end allows end-users to interact with blockchain Hyper-ledger and back-end system [S3], [S12]. In section 5.2.2, we further discuss the role of architecture-centric approach in BBS design phase.

5.1.4   Pattern-based BBS development

A software pattern represents a *general solution to a common recurring problem, one from which a specific solution may be derived* [60]. Patterns, as a timeless piece of reusable knowledge, are useful to resolve repeated problems faced during development for multiple blockchain platforms. Zhang et al., [S35] contend that due to the structural similarity of smart contracts to object-oriented program languages like C++, in terms of support for inheritance, abstraction, and polymorphism, many foundational software patterns can be employed to address the design challenges of smart contracts such as interoperability and reuse. Most patterns so far used for BBS, as discussed in Section 5.2.2, are attuned with the design phase and they are classified into three themes:

(i) existing well-known design patterns in conventional software engineering that are applied to resolve BBS development challenges such as *interoperability patterns* proposed by Zhang et al. [S35];

(ii) emerging platform-agnostic patterns that are results of blockchain practitioners' experience gained while developing BBS in real-world case scenarios such as Solidity coding practices [S36], *smart contract design patterns* by Liu et al. [S37], *blockchain data migration patterns* by Bandara [S38]; and *gas consumption*

(iii)   platform-specific patterns such as *Ethereum Solidity smart contract security design patterns* [S39], [S40]

For example, the collection of 15 patterns proposed by Liu [S37] have been identified and abstracted from different real-world BBS development scenarios. Bartoletti and Pompianu [S40] analyzed 811 Solidity smart contract source codes on the blockchain explorer (https://etherscan.io) and extracted design patterns related to transaction execution time management by a smart contract, i.e., *time-constraint patterns*, and encoding smart contract business logic execution to protect from critical operations, i.e., *math patterns*, and so on.

A unique and typical challenge of smart contract design is the gas cost consumption for the execution of transactions on blockchain platforms. Each transaction that is executed by smart contract will be charged some amount of gas, so the more intensive the computations are, the more gas fee should be paid for. The execution of a smart contract may be failed if the transaction is excessive in gas consumption. On the other hand, an accurate estimation of gas consumption by a smart contract may not be easy. From this point of view, design patterns and recommendations [S36], [S39] can aid software teams for smart contract design, with a view to save gas.

From these pattern collections, it appears that, although software patterns have gone some length to effectively address the problems of BBS design phase, their application to requirements elicitation and analysis are not yet explored in practice. Patterns related to requirements engineering enable software teams to capture right requirements, whose realization via BBS create added values, is an important avenue for future research. Another matter related to the deficiency of literature in adopting patterns for blockchain engineering is that almost all proposed patterns contribute to better BBS design. Despite their usefulness, there is a lack of patterns for specifying BBS development processes, which technically are referred to as *process patterns*, a term coined by Coplien in his landmark paper in 1994 [60]. Process patterns are the result of applying abstraction to recurring software development processes, thereby providing a foundation for constructing a bespoke software development process through the composition of appropriate process pattern instances. Applied in BBS engineering, process patterns are an invaluable source of insight for blockchain researchers and practitioners because they can normally reflect the state of the BBS development processes and are based on recurrent series of actions and well-established concepts.

5.1.5   Ontology-based BBS development

Ontologies help reduce conceptual ambiguities and inconsistencies in a particular domain while enabling value-creation capabilities [61]. The application of ontologies becomes important to facilitate knowledge interoperability among stakeholders. Ontology-based system development is initiated by identifying concepts/classes in a domain, and followed by assigning properties for each concept and defining domain constraints and relationships among these concepts that need to be verified if constraints are violated by these properties [62]. In view of the identified studies, the purpose of ontology-based BBS development is for training [S41], identification, design, and test of smart contracts [S42],[S43], and improving discoverability of smart contract services at the run-time [S44]. Kruijff and Weigand [S41] discuss the support of ontologies for BBS development by providing essential concepts related to an

operational BBS. Their proposed ontology can be viewed as a training material for software teams who might not be familiar with the key concepts of blockchain. This pioneering effort, in turn, has motivated others to explore alternative applications of ontologies to support the development activities of BBS, in particular the design phase, as will be delineated in Section 5.2.

5.2 RQ2: What are the key emerging and changing tasks in developing BBS?

BBS engineering aims at achieving business outcomes such as lower transaction cost, reduced dependency on trusted third parties, regulation compliance and certification, and secure data access. To this end, a series of tasks needed to be accomplished these outcomes. We identified these tasks from our primary studies and these are depicted in Figure 4. The tasks are grouped into four typical SDLC phases.
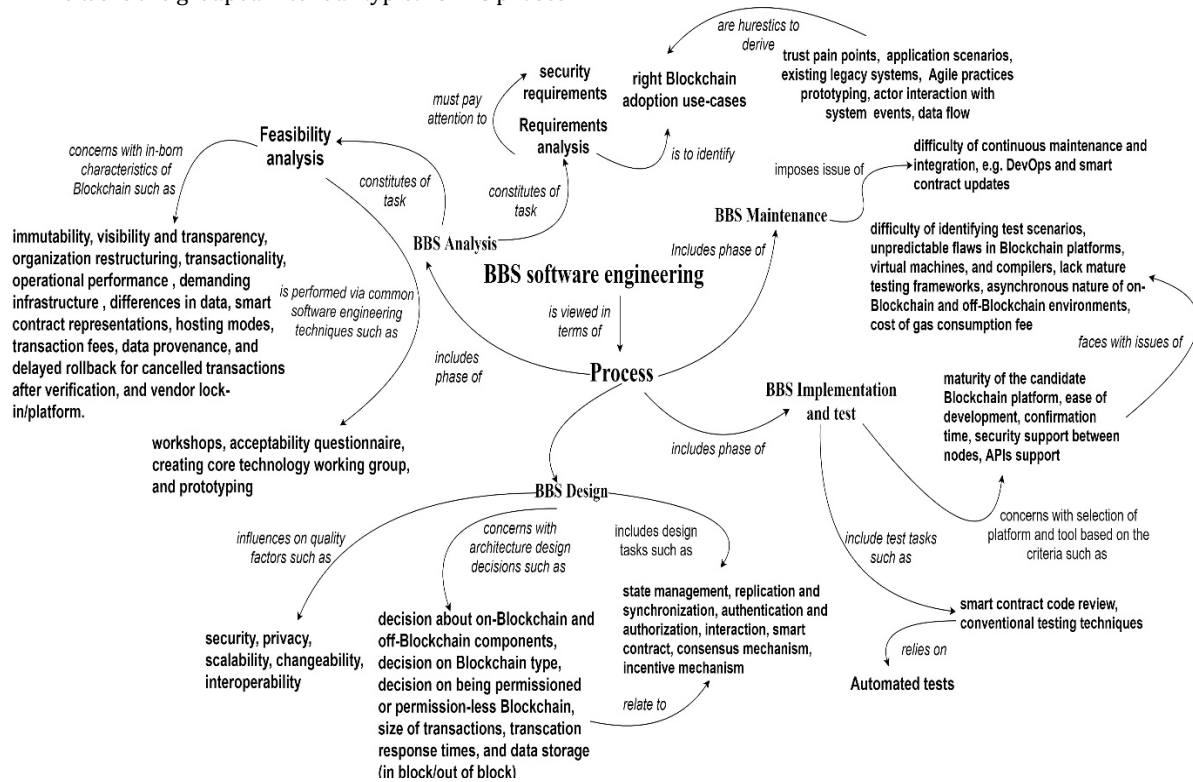

Fig. 4. *Process* conceptual map

5.2.1 BBS analysis

The analysis phase encompasses tasks related to requirements elicitation and verification. It establishes a link between stakeholders' needs and desired features of the target BBS. Note that tasks *feasibility analysis of BBS* and *requirements analysis of BBS* in our framework are by no means a complete description of comprehensive analysis phase of BBS engineering. They do highlight some distinctions of BBS development from non-blockchain software engineering. We elaborate on major challenges related to this phase as follows.

   Feasibility analysis of BBS. BBS is not merely a technical shift and a subject of regulation but also a substantial change in the way all stakeholders and systems hereinafter will operate, provide services to users, and be maintained. A BBS might be at the same time disruptive to some stakeholders and sustaining to others. Hence, the first essential task in the development process is to examine if blockchain technology enablement is a suitable option to embed into organizational business processes and to come up with an outline plan for the next phases. The most recent literature enumerates important *situation factors* (or *competing factors*) that can inhibit BBS development process and thus should be examined in the *feasibility analysis* task as shown in Table 3.

Table 3. Sample list of *situational factors* for consideration in *feasibility analysis of BBS* task

Category	Situational factor	Possible values for the factor	References
Organization	Management commitment	Low, medium, high	[S16],[S24],[S46],[S47]
	Organizational restructuring	Hierarchical, functional, horizontal, divisional, matrix, team-based, network	[S16],[S24],[S46],[S47]
	Business change	Rarely, occasionally, frequent, always	[S22], [S23], [S24]
Software team	Energy consumption cost	Low, medium, high	[S1]
	Development skills in cryptography, law/legislation	Low, medium, high	[S1],[S6],[S9],[S13],[S16], [S22],[S45],[S46],[S52], [S55],[S56]
Technical	Immutability of data	Stable, volatile	[S24]
	Data format	Homogeneous, mixture, substance	[S38]
	Visibility/transparency	Public, private, protected, package private	[S24],[S45]
	Transaction fee	Free, low cost, variable, expensive	[S38]
	Data provenance	Source, temporal, meta-data result	[S38]
	Transactionality	Low, medium, high	[S24]
	transaction performance	Critical, fairly important, low, moderate	[S24],[S25],[S26],[S47]
	Roll-back performance	Critical, fairly important, low, moderate	[S24]
	Demanding infrastructure	None, emergent, operating, matured	[S24]
	Vendor lock-in	Impossible, unlikely, even chance, certain	[S48]
	Hosting modes	Cloud, dedicated servers, virtual private,	[S38]
	Smart contract tool availability	Yes, no	[S31], [S52]

—*Immutability* achieved through cryptographic algorithms, can be a cost barrier to a BBS project [S24]. That is, if the need for the data persistency, immutability, and records' longevity is not greatly important, BBS may be an expensive choice, as implementation effort will be required in subsequent phases, in comparison to other data persistency mechanisms such as normative database management systems (e.g., Relational Database Management Systems). Stakeholders need to decide if immutability is a real need to be addressed by a BBS development instance.

—*Visibility and transparency* indicate that BBS enables all participants in the network to observe the chain of blocks, regardless of privacy and anonymity applied to the data, download blocks, and check them against contract rules. For example, participants can validate if there is a certain threshold for ordering items from a specific supplier, automatically via smart contracts. Visibility to external actors and systems is a situational factor that should be taken into account in the BBS project feasibility analysis task [S3],[S24].

—*Organization restructuring* means that BBS may raise complexities and changes in intermediary trusted legacy systems, cooperating organizations, boundaries, and layered based organizational structure for the coordination and enforcement of business legislations and privacy rules. This often results in a decentralized structure comprising of peers working according to fully transparent rules which are written in smart contracts [S16], [S24], [S46], [S47]. Hence, smart contracts should be adopted only if all parties agree to perform transactions on distributed ledgers. Findings in the research by Du et al., [S47] highlight that an incremental approach takes precedence over a revolutionary approach for organizations to develop BBS to accumulate tangible outcomes of addressing stakeholders' concerns prior to proceeding towards phases. Moreover, BBS presents a risk of software maintenance team downsizing. That is, as BBS increases the automation of decision making such as documentation, monitoring, verification, and auditing reports that are traditionally manually performed, staff associated with these activities will no longer be needed.

—*Transactionality* of organizational business services needs to be evaluated before converting them to smart contracts. If the type of business services is not concerned with verifying and enforcing credible processing transactions and data exchange between permitted participants, then BBS development is not worth it. For example, blockchain may not make an applicable case for simulation systems where the credibility of experimental data is not concerned. In such a situation, Scriber et al., [S24] suggest that non-transactional services may indirectly benefit from BBS via intermediator/proxy software components – depending on the definition of use-case scenario- where the intermediator sends/receives transactions between BBS and non-transactional services.

—*Operational performance* relates to the result of integrating existing legacy software with BBS. This evaluation should be based on factors such as acceptable latency and level of transaction verification. Unlike non-BBS systems, in which a transaction verification can be determined instantly, BBS harness is attributed by limited operational performance due to the latency of associated Proof of Work (PoW) algorithms to reach consensus and interoperability amongst chains/ledgers as pinpointed by findings of [S24], [S25], [S26], [S47]. A node might accept a transaction, but

wait for an extended period, based on the defined consensus model, for verifying, sharing, accepting, and encoding a transaction into the chain of blocks. If the target BBS is expected for a real-time response or specific performance, i.e. processing millions of transactions continuously, BBS will affect the overall performance and scalability [3]. To alleviate this issue, Du et al., [S47] suggest that software teams should, in collaboration with stakeholders, identify an essential dataset from business transactions for storing on blockchain and keep non-essential transactional data on traditional database systems. Furthermore, data and transaction security enforced by cryptography and PoW algorithms affect expected BBS performance. In particular, for IoT-based BBS, constrained devices might not be able to perform more advanced cryptographic techniques due to limited bandwidth, battery capacity, memory, processing power, and heat management issues [S24].

—*Demanding infrastructure (or technical maturity)* refers to providing support for the high degree of transaction processing, computational power to perform cryptography algorithms, storage size, and high bandwidth for distributed nodes, which are essential to deploy and maintain BBS. Organisations in poor regions or rural areas may not be able to afford these costs if are to set up BBS on their own infrastructural [S24].

—There are other concerns that cause BBS to turn out ill-suited and far-reaching consequences in later phases that turn out to be costly to rectify. These concerns, mainly related to the current technical limitations, are *gas consumption cost* (computational effort required to execute operations by smart contract) [S1], *development skills in cryptography* [S1],[S6],[S9],[S13],[S16],[S22],[S45],[S46],[S52],[S55],[S56], *business change* [S22],[S23],[S24], *delayed rollback for cancelled transactions after verification* [S24], *differences in data format*, *hosting modes*, *transaction fees*, *data provenance* [S38], *vendor lock-in/platform* [S48], and *tool availability for smart contracts test* [S31],[S52].

Software teams can embed the existing conventional techniques into the development process to conduct a *feasibility analysis of BBS* task such as conducting *workshops* [S31], [S49], *acceptability questionnaire* [S5], creating *core technology working group* [S50], and *prototyping* [S51] to evaluate if BBS simplifies the business processes and is an appropriate fit. It should be noted that the abovementioned factors may influence each other and hence should be investigated before moving to further phases of the BBS development process.

Requirements analysis of BBS. A highlighted challenge of requirements analysis in the context of BBS development is to identify right and cogent use-cases. Plansky et al., [S50] propose a four-step technique starting with the compilation of a pilot project that all stakeholders believe a distributed ledger can contribute to strong prospects. Existing pain points such as the lack of trust in the current collaborative business processes, conformance checks, intermediary delays, and areas of user dissatisfaction should be identified. Explicit hypotheses on how BBS improves these issues in terms of decreasing a certain level of organizational cost in a specified time period are defined in the second step. This is followed by developing the BBS prototype, for example, a smart contract, to attest hypotheses in the third step. If the prototype gets a clear sense of hypotheses acceptance, a project plan aiming at a few long-term goals e.g. increased revenue, better compliance, cost reductions, is set for the scaling up prototype in a measurable way. In the same vein, Fridgen et al., [S52] propose a technique with application examples in five idiosyncratic industrial cases. The technique consists of the following steps: (i) understanding blockchain technology to get a conceptual and technical foundation, (ii) deriving BBS application scenarios, (iii) analyzing existing legacy systems, (iv) documenting application scenarios, and (v) and developing and evaluating the prototype. Moreover, the requirements analysis in the proposed development process by Marchesi et al., [S3] relies on Agile practices and includes the following steps: (i) specifying blockchain adoption goals visible to stakeholders, (ii) identification of actors interacting with target BBS, e.g. human, external system, and IoT devices, (iii) determining the trust requirements between actors and BBS components, and (iv) documenting requirements in the form of user stories and expected features. Alternatively, the approach of Du et al., [S47] underlines the role of blockchain as a new *automated trust-building machine*. Accordingly, software teams should identify user inputs, manual verification, and reconciliation in transactions as candidate areas for converting them to smart contracts that create self-executing transactions through if-then conditions. A particular group of requirements that need more attention is related to security, which should be captured at the early stage of the development and be addressed in the design and test phases [S1]. Other than that, the literature suggests further techniques for requirements identification such as *events occurrence identification* [S48] and *data-driven* [S53]. They specify data elements of a problem domain and uncover data flow that is needed to be transformed and managed by smart contracts, respectively.

5.2.2 BBS design

The basic assumption in the design phase is that it leverages the output produced by the analysis phase (Section 5.2.1). The design phase dedicates identifying features to be exhibited and requirements to be satisfied by a specific architectural blueprint of target BBS. According to the identified papers, discussion on the design phase is organized into three essential architectural design decisions including (i) *decision on-blockchain and off-blockchain components,* (ii) *decision on blockchain type,* and (iii) *decision on being permission-based or permission-less blockchain*, as well as seven key design tasks: (i) *state management design*, (ii) *replication and synchronization design*, (iii) *authentication and authorization design*, (iv) *interaction design*, (v) *smart contract design*, (vi) *consensus mechanism design*, and (vii) *incentive mechanism design*. Both key architectural design decisions and tasks should be centered on the common system quality factors such as privacy, security, energy efficiency, scalability, and interoperability. They are realized in the next phase by implementing new software and hardware BBS components.

   Key architectural design decisions. The architecture of a BBS can be viewed as the set of interrelated design decisions that influence the quality factor of the overall BBS functionality. Thus, decisions are thus indispensable to the success of BBS development endeavor. These include:

—*Decision about on-blockchain and off-blockchain components.* This concerns determining functionalities that should be kept in a local organizational network and functionalities that are deployed in a blockchain network [S54]. Such segregation between on-blockchain and off-blockchain components, as used in [S1],[S11],[S30],[S38],[S39], and [S54], facilitates the separation of concerns and manageability of implementation and maintenance. On-blockchain components are smart contracts that are deployed and executed on blockchain platforms whereas off-blockchain components host off-line data and application business logic. Off-blockchain components interact with On-blockchain components via transactions. The interaction between these two groups of components is enabled via *mediator components* [S30] or *gateway connector components* [S54] that provide services for communication (transferring data), coordination (transferring control), conversion (adjusting unmatched interactions), and facilitation (optimizing interactions), as depicted in Figure 5.
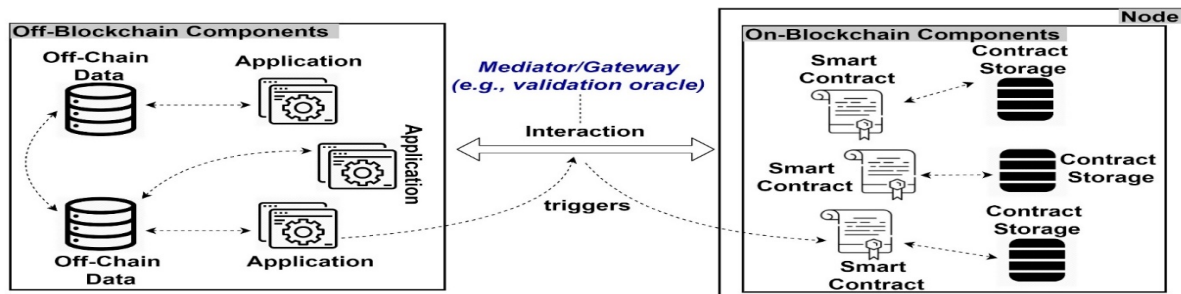

Fig. 5. A BBS architecture representation based on off-blockchain and on-blockchain components

—*Decision on blockchain type.* The type of blockchain network - public, private, and consortium- leads to making a trade-off between quality factors such as privacy, performance, traceability, and transparency. Unlike private and consortium blockchains where a single or a limited number of organizations authorize and control transaction validations, a public blockchain may include a large number of nodes. All committed transactions in a chain of blocks are visible, downloadable, and verifiable to all nodes in the blockchain network. Whilst this enables better information transparency and audit-ability, on the other hand, the transparency in public networks is at the expense of information privacy and scalability degradation for a large chain of blocks. Another example of trade-off architecture decision-making situation is that both private and consortium types offer partial immutability because participants can agree within the network to change, update, or delete a previous record. Despite the immutability constraints, private and consortium blockchain have significantly higher transaction throughput compared to public blockchain as well as cheaper set-up cost due to less required resources. Lin et al. [S1] recommend that public blockchain should limit the number of smart contracts and writing of permanent data due to the high costs such as computational power and data capacity involved in running computations and recording in the permission-less network. Therefore, software teams should not move all computations and data to the blockchain. A common recommendation, as jointly pointed by [S1], [S3], [S47], [S54], is to keep the big and private raw data off-blockchain whilst the tamper-resistant,

17

transparent, and traceable data reside on-blockchain. For instance, the hash of personal data can be stored on-blockchain and the raw data are saved to off-chain to provide better scalability and higher efficiency.

—*Decision on being permission-based or permission-less blockchain.* This concerns with the extent to which a participant plays in a blockchain network. The authority of permission management is granted by a permission-based blockchain network. Permission definition ranges from joining the network, transaction submission, transaction validation, asset creation to mining operations. For example, a participant node that is allowed to join the network has the read permission on the recoded transactions in blockchain. In a permission-less architecture, e.g., Bitcoin and Ethereum, a participant can be either pseudonymous or anonymous. In a permission-based architecture, e.g., Ripple and Eris Industries, the identity of participants is identifiable, which is similar to conventional commercial bank accounts . Hence, in permissioned architecture, participants need a legal identity to validate transactions. Both public and permission-less blockchain architecture do not guarantee data security and privacy. In other words, any node can join the network and visit all data records of participants. Since legal contracts that are executed by smart contracts will be accessible by participants in the public blockchain, software teams may be reluctant to use BBS or they need to select a permissioned blockchain architecture to grant or revoke permissions to participants. The permissioned blockchain can enforce access control to participant nodes. Whilst developers might want to encrypt transaction data on public and permission-less blockchain through cryptography algorithms to alleviate the privacy issue, on the other hand, a side effect of the encryption is the overall BBS performance and throughput degradation due to processing time needed for the data encryption/decryption. Hence, in the selection between permissioned or permission-less architecture for BBS, software teams should make the trade-offs between factors such as security, transaction processing rate, cost, censorship, and reversibility [S55].

—There are other design decisions listed by Arthur [S48] that influence architecture design, such as the *size of transactions*, *speed of response times*, and *data storage (in block/out of block)*.

In the following, we review seven essential design tasks that are frequently highlighted by our selected papers. We view the necessity of these tasks in relation to the layers of blockchain architecture as shown in Table 4. Based on the proposed layers in [S3], [S11], [S12], [S31], [S33], the following distinct layers of BBS are notable:

—*Application layer* packages existing legacy systems or new software applications and use cases interacting with smart contracts;

—*Smart contract layer* includes various scripts and algorithms that execute certain operations on data, memory, business process, and assets stored in blockchain network once pre-conditions are met.

—*Incentive layer* defines mechanisms for economic rewards in blockchain network to motivate participants to continue their effort for new block validation once it is created.

—*Consensus layer* contains possible consensus algorithms to reach transaction validity once it is added to the blockchain.

—*Network layer* specifies mechanisms of distributed networking for data forwarding, routing, and verification.

—*Data layer* stores the chain of data blocks via related mechanisms such as asymmetric encryption, time-stamping, and hash algorithms; and

—*Physical layer* includes hardware components such as servers, networks, and IoT devices on which BBS components are deployed.

==Figure to be added==

In performing these tasks, software teams face trade-offs among many quality factors across different layers of BBS architecture. These are explained along with illustrative examples in the following.

Table 4. Design tasks based on their correspondence to blockchain architecture layers

Task / Layer	State management	Replication and synchronization	Authentication and authorization	Interaction	Smart contract	Consensus mechanism	Incentive mechanism
Application	×	×	√	√	×	×	×
Smart contract	√	×	√	√	√	×	×
Incentive	×	×	×	√	×	×	√
Consensus	×	√	√	√	×	√	×
Network	×	×	√	√	×	×	×
Data	√	√	√	√	×	×	×
Physical	×	×	√	√	×	×	×

State management design. In an on-blockchain environment, a *state* is a snapshot of distributed ledger at a specific time. The state changes as a result of creating, updating, or deleting blocks. In such a self-contained environment, a ledger keeps a full record of state transitions in a timestamp-sequenced, immutable, and tamper-resistant way. In an off-blockchain environment, however, external software applications do not keep these states to perform their transactions. This means the state of external systems is not accessible to on-blockchain components, for example, a smart contract, if they need to know that state to complete its functions (Figure 5). This can be problematic for the validation of cross-domain transactions that may depend on the states of both on-blockchain and off-blockchain environments. To handle this issue, some useful design patterns, are proposed by Xu et al., [S26] and Zhang et al. [S35]. These are concerned with *interaction with the external world* and resolve the interaction issues between BBS components and the external components [S26], [S35]. The patterns are suggested to design state management components and they are based on the notion of (i) *Oracle*, i.e., storing the state of external systems into the blockchain execution environment and (ii) *reverse Oracle*, i.e., creating an interface between external systems and smart contracts for supplying data and check conditions, and (iii) *validation oracle* i.e., a state coordination mechanism to check conditions that cannot be represented in an on-blockchain environment. If a transaction validation depends on external states, the validation oracle is asked to validate and sign the transaction. The validation oracle mechanism can be performed by automated or human arbitrator by injecting external states into blockchain via regularly updating the values of external application states within the contract storage. Alternatively, Luu et al. [S17] propose a technique in which a transaction T defines a guard condition g too. The current state should satisfy g for the execution of T to proceed. If g is not satisfied, the transaction is simply dropped by the new rule. For transactions which do not provide g, we simply consider $g \equiv$ true. This solution guarantees that either the sender gets the expected output or the transaction fails. Nevertheless, software teams should be aware that all these ways affect the execution performance of transactions and cause a delay due to the need to make sure the consistency of states between on-blockchain and off-blockchain components by *validation oracle* mechanism [S55].

Replication and synchronization design. There are differences in the design of data replication and synchronization between conventional software systems and BBS [S55]. In the non-BBS context, software teams adopt common data partitioning and replicating mechanisms such as master-slave or multi-master replication to improve the throughput, sustainability, and latency of a software system. In contrast, in BBS harness, each data block is duplicated over network nodes, which causes unwanted increase in latency and reduction of throughput. Moreover, for the data synchronization design in conventional software architecture design, there are well-known mechanisms such as 2-Phase Commit and Paxos to keep data replication synchronized. The synchronization mechanism in BBS is based on reaching consensus among network nodes to perform correct operations regardless of faulty components. Additionally, consistency control on performing transactions in BBS is based on the defined terms and rules in their associated smart contracts. The choice of replication and synchronization mechanisms [13],[18],[19],[37], will be, unavoidably, influencing final BBS quality factors of throughput, sustainability, and latency.

Authentication and authorization design. Specific mechanisms should be designed to enable the identity detection of an entity in a blockchain network, which either manages data for operational/business purposes or provides services to participants, via cryptography techniques such as digital signatures, homomorphic encryption, and multi-party computation [13],[18],[19]. In particular, for a permissioned blockchain, an additional access control layer should be defined via Certificate Authority (CA) and a Membership Service Provider (MSP). Hebert et al., [S49] list a comprehensive set of security risks including fifteen general categories of risks from key management to privacy, consensus, interoperability, and legal concerns. To tackle these risks, [S1] and [S3], jointly, define a set of security design principles that should be realized via implementing new components in BBS architecture. The following sub-activities are recommended:

—*Log design.* BBS reflects historical records and the usage of data by any entities such as who, why, when, what and how, that all should be logged in the log ledger in a verifiable way.

—*Policy design for data usage.* The policy indicates rules for topics such as access rights, permissions, and conditions.

—*Off-Chain data storage design.* Storing sensitive data directly in blockchain, regardless of encryption techniques applied, can be at the risk of privacy leakage. Thus, the sensitive data should be stored in off-blockchain components such as conventional database management systems (e.g., Oracle or MongoDB), cloud storages (e.g., S3, AWS or Azure), and distributed storages (e.g., IPFS or Storj). The reference to these data should be stored in

the blockchain. Such a reference can be a hash, a connection string, an absolute path, or an identifier referring to a dataset.

—*Circuit breaker design*. This indicates necessary emergency termination functions in smart contracts that are triggered in the case of bugs or the occurrence of abnormal behaviour by an entity.

Interaction design. This task is to specify interdependencies, integration pain points, and the way BBS components can communicate, exchange, and use each other's services. It helps characterize the overall architecture and the position of BBS in the operational environment. A key concern during the interaction design task is the interoperability factor across the BBS layers. This is due to reasons like (i) switching BBS between single or multiple blockchain platforms, (ii) BBS integration with existing legacy software applications, and (iii) multiple types of BBS support the completion of a cross-domain transaction within networks, which involve computational resources owned by different entities, (iv) changing hardware components on which BBS instance is running, and (vi) changing a smart contract to a new one or combing several smart contracts into a single one. For instance, some smart contracts that are upgraded to new Ethereum Virtual Machine (EVM) with the aim of strengthening the security and optimizing performance, may raise interoperability issues – in the scope of the platform versions, smart contract languages, and APIs. Hardjono et al., [S56] investigate the interoperability issues in BBS architecture design and propose the following root cause analysis to identify interoperability pain points: (i) *minimal assumption*, i.e. identifying the unit of transaction that is semantically recognizable for multiple BBS, (ii) *degrees of permissionability*, i.e. identifying transactions that are processed by both permissioned network and external permission-less networks (foreign domains), and (iii) *participants*, i.e., identifying network's nodes that are involved in transaction processing. In addressing the interoperability pain points for cross-domain transactions, Hardjono et al., [S56] rely on the notion of peering agreement in autonomous systems and recommend steps of designing (i) semantic compatibility that is required for multiple BBS, (ii) data exchange and communication protocols, (iii) mediators, delegations, and technical-trust mechanisms, and (iv) agreements such as level of services, fees, penalties, liabilities, and warranties.

Recall from Section 5.1, software patterns can be used to resolve interoperability issues between on-blockchain and off-blockchain components. Zhang et al., [S35] exploit the application of software design patterns to address BBS interoperability design issues and choose e-health software applications as a representative use case. The application of four sample software patterns, namely *abstract factory*, *flyweight*, *proxy*, and *publisher-subscriber* is appraised. For example, to abstract away the implementation detail of data storage of a smart contract, the proxy pattern is suggested that enables seamless interaction between BBS components while still supporting variations in smart contract's data storage options. Moreover, the abstract factory pattern provides the creation services to instantiate generic smart contracts to domain-specific ones. Heterogeneity of blockchian platforms and legacy systems and support for the data migration from on-premise to blockchian platforms or migration between multiple blockchain platforms have been taken into account from software pattern-oriented view by providing *blockchain data migration patterns* [S38] such as *state extraction pattern*, *state transformation patterns*, *state and transaction load patterns*, and *safe data management*.

Smart contract design. The aim of this task is to (i) analyze existing textual legal contracts, (ii) identify and convert semantic clauses and rules that are expressed as smart contracts, and (iii) transform smart contracts expressible as software codes. Derived from our selected papers, we broadly divide the task of smart contract design into two sub-tasks of *(preliminary) high-level design* and *(detailed) low-level design* as shown in Figure 6.
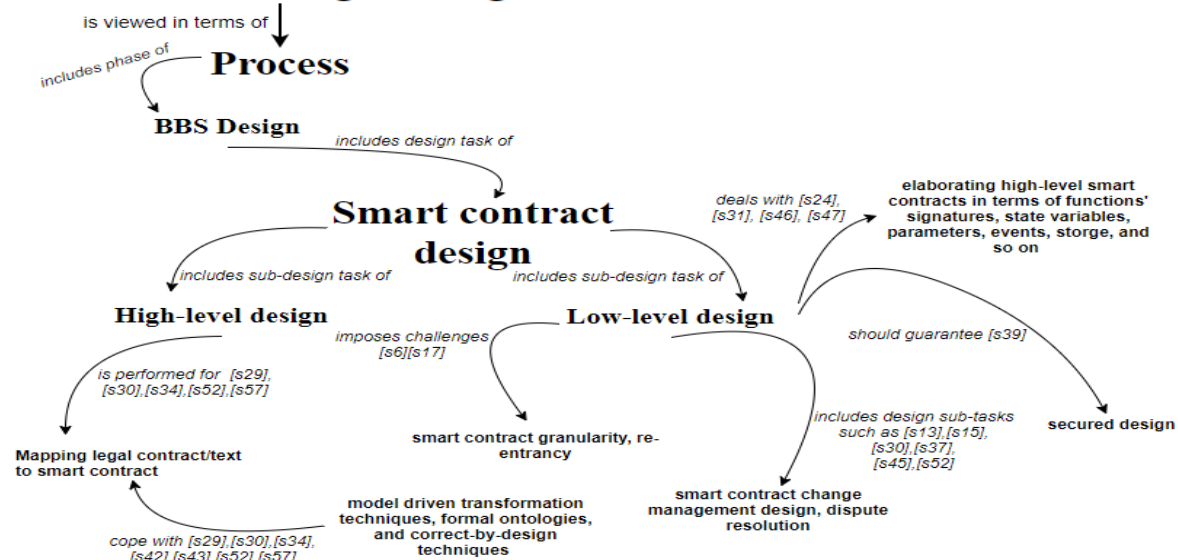
Fig. 6. *Smart contract design* conceptual map

The *high-level design* sub-task is concerned with crafting an initial structure for smart contracts by translating clauses of legal (textual) contracts, business transaction logics, or backend codes into smart contract codes, e.g., Ethereum Solidity scripts that are executable on blockchain platforms. A key challenge in smart contract design is how to map legal contracts to executable software code in a correct way. There are important differences between the content of a legal contract expressed in the natural language and a corresponding smart contract code. Some prose of a legal contract may not have equivalent expressions in a smart contract and vice versa. Moreover, a legal contract may contain certain terms like a reasonable excuse, unintended delay, exempted penalty, and so on that are deemed implicit in a natural language. However, they may have no explicit meaning at the level of smart contract code. Hence, in transforming contract documents and legal prose to equivalent smart contracts, software teams encounter ambiguities and different interpretations that can be reflected in smart contract code.

As briefly introduced in Section 5.1.1, using ontologies and formal/mathematic expressions to describe legal text can potentially reduce the transformation errors in legal contract to smart contract mapping and to ensure legal prose that are expected to meet. However, this needs training effort and acceptance by all BBS project communities (software teams and stakeholders) to use formal constructs to describe legal contracts. Domain ontologies are suggested to enhance the identification of target smart contracts including their functions' specifications, relationships, automated inference, and verification. Kim and Laskowski in [S42] use first-order logic to express the ontology of smart contracts that later execute a provenance trace and to enforce traceability constraints on transaction execution on blockchain. Following this introductory work, Choudhury et al. [S43] introduce a framework that uses semantic rules to encode BBS functional requirements and to leverage the structure of abstract syntax trees for the automatic generation of smart contracts. Also, ontology design techniques [61] enable developers to elicit potential network participants in business processes. They also help identify participants' relationships, conditions, and constraints in terms of resources exchanged between themselves which can be, accordingly, represented by smart contracts. Furthermore, ontologies are also proposed to improve the discoverability of smart contract services that are offered by a deployed BBS. The notion of *semantic smart contract* is proposed by Baqa et al. [S44] to use RESTful semantic web to enable indexing, browsing, and annotating of smart contracts. As such, annotated smart contracts deploying on distributed ledgers can be treated as linked data for performing queries by domain-specific terms, which increases the discovery of public smart contracts.

In addition to the application of ontology design techniques in the smart contract design, recall from Section 5.1.2, MDD can play a key role in the design phase of BBS engineering, in particular, in reducing mapping complexity, i.e. enforcing control-flow and conformance to normative business processes via smart contracts [S29],[S30]. Frantz and Nowostawski [S57], propose a semi-automated technique for transforming human-readable behavior specifications

to a smart contract format. They borrow the idea of *institutional analysis* from the social and economic field in order to analyze and decompose the functions of human institutions into a set of declarative rule-based statements. Statements are expressed via core elements such as attributes, actions, outcomes, conditions, and consequences associated with non-conformance. The mapping of core constructs, which is automated via a domain-specific language (DSL) such as Scala, provides a foundation to transform rule-based statements into contracts. An alternative design technique is proposed by Clack et al., [S52]. It is based on the idea of separation of concerns through which three levels of abstractions for a smart contract are defined including: *legal contracts*, *smart contract templates*, and *smart contract codes*. In this layered-based contract design, a legal contract includes text (a linear sequence of sentences, hierarchical structure) in the natural language, lists/tables, cross-references, and redacted text (privileged and proprietary text). A smart contract template is an intermediate and electronic representation of the legal contract documents. A template contains different parameters such as identity, data types, and values. It is developed by standard bodies and uses parameters in legal documentations to connect it to corresponding codes. All stakeholders should have agreements on these parameters and their values as they directly influence their business relationship as well as the way codes in smart contracts will operate. While a smart contract template tries to formalize actions, legal rights, and enforceable obligations that accrue to different actors and often are context-specific, a smart contract is an automated version of the smart legal template by a software script. This harness forms the layers of transformation which can be performed in a semi-automatic way using appropriate tools. A similar technique for high-level smart contract design, named *correct-by-design development*, is proposed by Mavridou et al. [S34]. The technique is based on an end-to-end formal model-based verification. It describes the security requirements via abstract transition models along with semantic operations and reasoning at the abstract model level to examine if the behavior of the contract models satisfies these requirements. The formal models are then transformed to Solidity code bytes for execution on Ethereum.

Smart contracts should provide mechanisms to guarantee the security and privacy of sensitive data access and process. Yet security challenges related to smart contract design are multitude. Although it is not in the scope of our survey to cover all the security-associated issues of smart contracts, there is a wealth of pattern collections, proposed by Wohrer et al. [S39] that deal with the security concerns in smart contract design. Software teams can adopt patterns such as *checks-effects-interaction*, *speed bump*, *rate limit*, *mutex*, and *balance limit* for safe and reliable execution of smart contracts.

Low-level design concerning the elaboration of high-level smart contract definition on functions' signature, parameters, contract storage, events, and state variables, the reviewed papers recommend performing the following alternative analysis to define custodial functions to support primitive functions of smart contracts:

—analyzing actors who interact with smart contracts of BBS ([S31], [S46], [S47]);

—analyzing interaction points and message flows between smart contracts and external components, e.g. smart contracts and legacy systems [S46];

—analyzing events where a BBS is viewed as a stimulus-response machine reacting to smart contracts and external components, e.g. smart contracts and legacy systems [S46];

—analyzing external behaviors of a BBS to identify states that a smart contract can enter or trigger events [S31], [S46];

—support of cancellation of transactions and delayed rollback [S24];

—specifying required libraries, data structure, inheritance structure, and interfaces for smart contracts [S31];

The low-level smart contract design also raises two common challenges that software teams should be aware of:

—*Granularity*. The granularity of a smart contract is defined in terms of the number/size of required processing functions in smart contracts to enforce rules and control the execution of business transactions [S6]. For large-scale transactions, an individual smart contract negatively affects power consumption, storage space, and performance. For instance, Ethereum charges a computation fee for smart contract execution. Likewise, smart contracts are performed sequentially on blockchain platforms, which reduces the overall performance of BBS if there are many smart contracts that are bounded to a business process. The parallel execution of smart contracts is a resolution technique to address this issue. However, a detailed discussion about the concurrent smart contract models is discussed in [13],[18],[19],[37], which falls out of the scope of this survey.

—*Re-entrancy*. This issue occurs in an interaction between two smart contracts when the control of smart contract execution is taken over from one to another during the execution of cross-functional transactions. Once a smart

contract calls another smart contract, its current execution is stopped until the other call is finished. The intermediate status of the first smart contract may not be ready or correct for other BBS components to use. This, for example, can be a critical issue associated with money withdrawal function. Smart contracts should define discrete, atomic, and sequential functions in a way that when a function is called, it cannot be re-entered by an external call before its current execution is entirely ended [S17].

Apart from the discussion on high-level and low-level smart contract design, the literature draws attention to sub-design tasks related to smart contracts:

—*Change management design*. As mentioned earlier, legal contracts may include complex conditions and terms that are updated based on agreed upcoming changes by parties. On the other hand, once a transaction is generated and stored by smart contracts on blockchain database, it is infeasible to modify this transaction as the blockchain is merely an append-only that supports the creation of a new transaction to add to the chain of blocks instead of updating already existing blocks. This is realized via defining change management function and control variables in smart contracts. This is in contradiction with evolving requirements and business environments. Unforeseen changes in legal prose and rules frequently occur after a contract document is finalized and signed by parties due to reasons such as a change in business goals or valid rescission by courts (no longer a cause of action for agreement breach) that yield in changing smart contract parameters and functions to meet new requirements. This is crucial to address change requests in smart contracts and to assess the change impact and propagation for keeping consistency between stakeholders' requirements and the design of smart contracts. Software patterns proposed by Weber et al. [S30] and Liu et al. [S37] open new possibilities for ameliorating smart contract change management. Weber et al. [S30] suggest using *contract factory* defining methods for instantiation of specific case contracts from predefined generic contracts. There are works in progress to handle the smart contract evolution, e.g. *delegation technique* where the data and logic of the contract are separated and *registry contract technique* which keeps the track of contract version and records the history of changes and rationales [S13],[S45]. Marino et al. [S15] suggest guidelines for smart contract change management that have been applied for altering and undoing smart contracts on Ethereum platform. The guidelines that are taken from legal contracts and adjusted to the context of smart contracts, briefly, are (i) modification by right resulting in a new contract, (ii) modification by agreement, and (iii) reformation. Despite this, change management mechanisms for smart contract have not been fully explored in the context of BBS engineering process, however, it is an important topic for future research.

—*Dispute resolution design.* The enforcement of smart contracts for business services raises the question about the scope and responsibilities in the case of arising disputations whether they should be resolved automatically by smart contracts or human intervention [S52]. In a business environment, a violation or illegal act against a contract can be resolved by a well-established body of laws such as imposing fines, asset detachment, and service deprivation in courts. On the other hand, despite some debates, violations like malicious security attacks, network disruption, and power cut at software or network levels cannot be fixed without the need for dispute resolution. Such issues imply that software teams should clarify and verify with stakeholders the boundary and scope of user stories and actor interactions that should be converted to smart contracts.

Consensus mechanism design. Validator nodes are responsible for validating a newly added block to the blockchain. The new block is propagated to the network and is appended to the chain of blocks once all validators reach a consensus to ensure that they have an exact copy of the new block. In this regard, the task of consensus mechanism design deals with defining an agreement mechanism satisfying all validators involved in the network when a new block is created. Efficient consensus creation has been an important query in distributed system engineering design and several consensus mechanisms have already been explored over the years. Some are transplantable to the BBS context. The choice of prevailing consensus mechanisms such as proof of work (PoW), delegated proof of stake (DPoS), proof of importance (POI), and proof of stake (PoS) [13],[18],[19],[37] requires a software team to make the trade-off between quality factors such as power consumption, scalability, and simplicity. The complexity of the consensus mechanism depends on the network type. A permissioned network needs less complicated mechanisms due to restricted network access and vulnerability to security attacks. On the flip side, in a permission-less network, an energy-intensive mechanism, such as PoW negatively affects BBS scalability and thus reduces the transaction processing throughput in contrast to permissioned networks that tend to utilize non-compute-intensive mechanisms like DPoS.

Incentive mechanism design. Due to the power consumption and computation fee charged in the network to validate new blocks of transactions, some economic rating and reputation mechanisms are defined for validators, e.g., miners.

The incentive mechanisms for active validators can be defined for different aspects such as a gas fee for transaction execution, storage price, security deposit, data retrieval, and so on [S54]. Additional economic incentives are needed for validators if BBS has computations that are being run off-blockchain.

5.2.3 BBS implementation and test

The realization of the designed architecture in the previous phase is achieved via developing software and hardware components. According to the identified papers, a key aspect in conjunction with the implementation and test is the choice of blockchain platform. While tool support for the whole BBS lifecycle automation may not be feasible, the review of selected studies reveals several popular ones, either open-source or commercial, which are used during this phase for on-blockchain components. The example of these, as shown in Table 5 and alphabetically sorted, are *BigChainDB*, *Chain Core*, *Corda*, *Credits*, *Domus Tower*, *Elements*, *Ethereum*, *HydraChain*, *Hyperledger Fabric*, *Hyperledger Iroha*, *Hyperledger Sawtooth Lake*, JUICE, *Multichain*, *Openchain*, *Quorum*, *Stellar*, *Symbiont Assembly*, and *Truffle*. Apparently, according to the reviewed studies, the most frequently used platforms are *Hyperledger Fabric* and *Ethereum*. For instance, Hyperledger Fabric [63] is an open-source development platform, hosted by the Linux Foundation. It acts as enabling technologies to model and integrate existing systems with blockchain platforms. Hyperledger Fabric's programming model, supported by technologies Node.js and Java, enables more intuitive development in a plug-and-play fashion. An analytical comparison of features offered by these platforms is out of the scope of this survey. However, it has been a research query in the work by Bettín-Díaz et al. [S18] synopsizing their experience in BBS engineering –with an application case in the supply chain. The authors recommend five key factors that software teams should investigate in their platform choice and, thereby, make an alignment between technical and managerial decisions. They are (i) *maturity* of the candidate blockchain platform in terms of how long it has been in the market, supporting model, and availability of documentation and training materials, (ii) *ease of development* which is based on required programming skills to work with the platform, (iii) *confirmation time* which depends on the choice of consensus mechanisms, (iv) *security support between nodes* as a platform may provide functionalities to enable public or private network configuration, and (vi) *APIs support* for functions related to audition, authentication, hash generating, data storage and retrieval, and smart contract lifecycle management. Furthermore, the literature suggests the use of mainstream tools and programming languages for coding, scripting, transferring/streaming, and data manipulation for off-blockchain components.

Table 5. Sample technologies used during the phase of BBS implementation and test

	Technology	Type	Aim
On-blockchain components implementation	BigChainDB	Blockchain platform	A big data distributed platform with a support of blockchain characteristics such as de-centralized control, immutability, and digital data transfer mechanisms
	Californium CoAP	Development framework	Securing data transfer between IoT device data and blockchain platforms
	Chain Core	Blockchain platform	Issuing and transferring financial assets on a permissioned blockchain infrastructure.
	Corda	Open source distributed ledger	Enabling the development of smart contracts with a support for pluggable consensus mechanisms and minimizing transaction cost
	Credit	Distributed ledger development framework	Developing permission based smart contracts
	Domus Tower	Blockchain platform	Implementing consortium blockchain with a focus on finance domain functionalities
	Elements	Open source blockchain platform	Enhancing Bitcoin functionalities at the communication and protocol levels
	Ethereum	Blockchain platform	Enhancing Bitcoin functionalities and implementing smart contracts.
	Eris:db	Distributed ledger	Enhancing Bitcoin functionalities
	HydraChain	Blockchain platform	An extension to Ethereum platform for creating permissioned, private, and consortium blockchain
	Hyperledger Fabric	Open source blockchain platform	Providing support for smart contract implementation and test in different application domains
	Hyperledger Iroha	Distributed ledger	Developing smart contracts for mobile-based applications

Hyperledger Sawtooth Lake	Open source blockchain platform	Providing support for smart contract development with specific support for decoupling transaction business logic from the consensus layer
JUICE	Tool	Enabling the implementation and monitoring of Solidity smart contracts running on Ethereum platform
Multichain	Open source blockchain platform	Supporting Bitcoin functionalities for multi-asset financial transactions
Openchain	Open source distributed ledger	Issuing and managing digital assets via smart contracts
Quorum	Distributed ledger	Developing smart contract platform based on Ethereum
Stellar	Distributed ledger	Enabling distributed payments infrastructure with RESTful HTTP API servers
Symbiont Assembly	Distributed ledger	A distributed ledger based on Apache Kafka to develop smart contracts
Truffle	Framework	Compilation, test, integration, and deployment of smart contracts
Bitcoin Testnet	Framework	Testing smart contracts without changing real system data or transactions
Hyperledger Besu	Open source development framework	A Java-based Ethereum client to develop and deploy applications to run on the public Ethereum public network or private permissioned network
Mininet	Tool	An emulator to analyse transaction blockchain transaction delays

Off-blockchain components implementation

JSON RPC	Protocol	Remote procedure call used by Ethereum clients to interact with Ethereum nodes
Web3j	Library	A lightweight Java and Android library for working with smart contracts and integrating with Ethereum platform with the minimum overhead for implementing integration codes
CouchDB	Database	A document based NoSQL database that uses JSON to store data, JavaScript as its query language, and commonly used with Hyperledger Fabric
Raspberry Pi	Toolkit	Collection of hardware and programming language to develop blockchain-based IoT systems
REST API	API	To query data and test HTTP requests as well as call blockchain platform APIs
Bluetooth, ZigBee, WiFi, 2G/3G/4G cellular	Hardware	Hardware Communication protocols
Apache Tomcat, Eclipse Photon and WebStorm	Platform	Hosting and implementing back-end and front-end applications interaction with off-blockchain components

Testing is a key engineering task to ensure that an implemented BBS satisfies the specified requirements from the analysis phase. BBS testing that takes into account factors such as type of participants, permissions, input/output states, predefined trigger conditions and response actions of smart contracts, transactions, and expected outcomes in testing scenarios, is very much comparable to the conventional software engineering testing and can be performed at three levels:

—*unit testing* that is to analyze each BBS component, for example, smart contract source code, to identify code segments prone to vulnerabilities during transaction execution and creating the chain of blocks;

—*integration testing* that is to verify if on-blockchain and off-blockchain components work together correctly; and

—*user acceptance testing* that is to validate the whole BBS.

Baqa et al. [S44] define a three-step strategy to conduct BBS testing:

—*test map* creates a map of on-blockchain and off-blockchain components for which test is required; and

—*test plan* specifies how each type of tests, such as unit and integration testing, should be performed within a planned number of test cases. For instance, for the performance test, software teams need to include parameters like measured execution time to record a transaction (time is spent to send a transaction request and receive confirmation from the receiver) and the time of cryptographic algorithm execution and consensus creation.

—*test run* performs test scenarios on a *testnet*, i.e., a testing environment, for example Bitcoin Testnet, allowing software teams to experiment BBS without worrying about real transactions or using bitcoins.

Conventional software engineering testing techniques such as code coverage review and stress testing may still have value and be applicable in BBS context, as acknowledged in [S1],[S44],[S52]. For example, a performance test principally is meant as the speed of adding a new block to the chain of blocks, and throughput of mining, consensus creation, and transaction validation. However, conventional software engineering testing practices appear to be insufficient on their own due to some intrinsic features of BBS. Testing smart contracts is crucial (Section 5.1 and 5.2.2). They are immutable and no one is supposed to tamper or change their source code once deployed and stored on a blockchain platform. Unlike conventional system development, further updates to fix identified bugs in smart contracts become an issue. Findings from a survey of 232 smart contract developers, conducted by Zou et al. [S13], highlight the top testing challenges of smart contracts as (i) difficulty of identifying test scenarios, (ii) unpredictable flaws in blockchain platforms, virtual machines, and compilers, (iii) the lack of mature testing frameworks compared to well-established languages like C++ and Java, (iv) asynchronous environments of on-blockchain and off-blockchain, and (v) cost of gas consumption fee for testing smart contracts. *Code review* is a basic way to ensure the validity of a smart contract and, to this end, software teams can employ techniques such as (i) peer smart contract code review, (ii) requesting blockchain practitioners from GitHub community to check a smart contract, and (iii) asking third-party reviewers to audit smart contract code. Zou et al. [S13], however, raise issues of code review including being time-consuming and difficult to find expert developers to identify security flaws in smart contracts. To lessen the complexity of code review, [S51] and [S52] similarly suggest three stages of testing in order to gradually identify and fix bugs in a smart contract code: (i) deploying contracts on a local network, (ii) deploying contracts on the test network for software team to use, and (iii) deploying contract on the live and main network and to execute them by users.

Automated testing techniques are essential to reduce the complexity of code review and to identify vulnerabilities of a smart contract before its deployment. In line with this, as mentioned earlier in Section 5.1.2, using MDD enables developers to represent the general features of a system at a high abstraction level without being constrained to platform-specific implementation details. A common technique to verify a smart contract is *model-checking*. Applied in BBS context, it enables the production of code coverage test scenarios to examine the correctness of function execution, sequence of functions, branches, and trigger points in smart contracts. Model-checking can be accommodated in the implementation phase for automatic generation of smart contract codes based on system's behavioral state models and non-trivial environmental interactions, e.g. [S31], and consistency checking between the smart contract deployment model and deployment scripts for distributed ledgers, e.g. [S32]. In this spirit, Destefanis et al., [S52] propose a model-based testing technique as a rigorous mechanism for automated smart contract tests where a set of particular fault-related test cases are generated from an abstract fault model. However, the test scalability of automated tests in a large number of smart contracts is a limitation.

5.2.4 BBS maintenance

As the majority of works focus on analysis, design, implementation, and test phases, the discussion on the maintenance-related tasks is hardly found in the current literature. This is, perhaps, due to the fact that, apart from the characteristics of BBS, the maintenance for BBS has no difference at its core with non-BBS ones. The only study that provides empirical findings on the maintenance phase is by Bosu et al., [S22], which highlights the issues that are not noticeable in non-BBS development. That is, as software teams can upgrade conventional software applications with new features as supported by core technologies like DevOps [52], BBS needs to wait for the preparation of all nodes around the network to upgrade new functionalities which may cause issues such as outdated results, delay in synchronization, and costly response to changes in the business domain. As mentioned earlier (Section 5.2.3), due to the difficulty of making changes in smart contracts after deployment, backward compatibility, the capability of earlier transaction validation, or roll back to a previous version of smart contracts are challenging.

5.3 RQ3: What software modeling approaches and notations are applicable in BBS development lifecycle?

Our perspective on BBS engineering here is divided into *models* and *modeling languages* (Figure 7). Models conceptualize different aspects of a BBS and *modelling languages* determines a means, i.e. syntax and semantic, to express these models.

5.3.1 Models (work-products/artefacts)

BBS engineering can be viewed as series of intermediate models that are generated and evolved to achieve a final BBS. The models are results of development process activities performed by software teams and they help trace how high-level stakeholders' requirements are transformed to executable BBS. The importance of models in BBS context is much comparable to conventional software development. Since models may be cost-prohibitive to generate, either manually or automatically, due to their evolution and maintenance cost, generating the models during the development process is situational and dependent on factors such as project requirements, the project domain, and developers' opinions, and hence their generation does not happen in a vacuum.

The literature suggests some notable models associated with the analysis phase. Amongst them, a *requirements model*, which has been frequently stated in the reviewed studies by different terms such as *user stories* and *goal statement* [S1], *use-case model* [S23],[S31], *collection of requirements* [S31], s*ystem requirements* [S46], and *business model* [S51], collectively, describes functional/non-functional requirements to be fulfilled by target BBS and form the basis for analyzing the remaining domain. Lin et al., [S1] in their experience of developing a security system for the telecommunication domain suggest documenting requirements as user stories where customers write a short description of a certain transaction that is needed to be delivered by target BBS in their own terminology and index cards. Marches et al., [S23] propose *use-case models* as a starting point to identify how actors will interact with and send/receive transactions to BBS. *Use-case models* should capture features and functions specific to processing regulations and technical requirements such as legal constraints and cryptographies. Requirements models are compiled in a feasibility report, supplemented by a primitive *prototype*, also referred to as *blockchain compliance checklist* [S51] and *theoretical build-up* [S55], to get a better understanding of requirements and the technical feasibility of BBS. As mentioned earlier, prototypes are not only meant for requirements elicitation but also provide a foundation for architecture models.
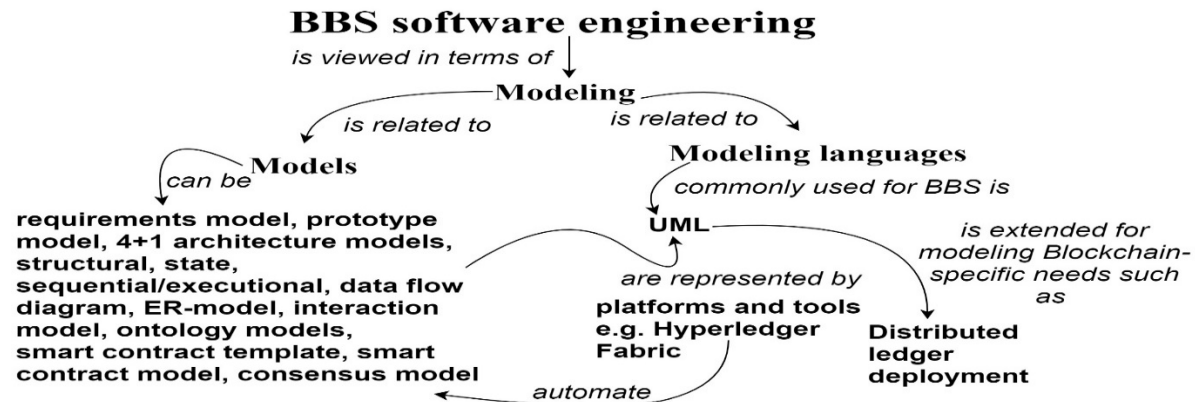

Fig. 7. *Modeling* conceptual map

The literature recommends some models related to the design phase of BBS engineering. Recall from 5.1.3, the set of well-known software engineering *architecture models*, 4+1 views [58], are used in the context of BBS design phase. They describe the structural relationship and interaction between software and hardware components of BBS. For instance, a *process view* of the architecture shows the sequence of events occurring when users or external systems call a smart contract functions (Figure 3). Similarly, the *deployment model*, one of the commonly recommended 4+1 models, as noted by Górski and Bednarski [S32], represents how (i) BBS off-blockchain components access on-blockchain components and (ii) BBS architecture is operationalized with respect to a chosen blockchain platform, physical layers, and installation scripts to make BBS available to end-users. Additionally, Rocha and Ducasse [S53] suggest the usefulness of entity-relationship models (ER-model) to represent conceptual and logical design for blockchain data exchange between on-blockchain and off-blockchain components.

Apart from the well-known models that are recommended above, software teams may generate some blockchain-specific models during the development process. A smart contract model, as recommended in [S1],[S5],[S31],[S46],[S56], [S57] represents the functionalities that realize requirements.

The rationale for the applicability of object-oriented modeling in BBS is that a smart contract model can be analogically viewed as a *Class* model in object-oriented programming. Due to different options to call their function,

smart contract models are classified into *structural*, *state*, and *sequential/executional*. These three diverse models are to represent the structure and relationships, transitions states, and sending/receiving messages from/to a smart contract, respectively. To be more specific, a structural model, akin to a class model, shows data, public, and private functions of a smart contract, and allows for reuse and inheritance from other smart contracts. A state model enables developers to identify the different statuses of the contract in its whole life cycle from the stage of deployment to sign-in, execution, and termination. As such, different changes in the status of a smart contract, e.g., chaining of transactions, as results of internal operations or interactions with external systems can be identified. The other useful model at the design phase is a sequence model. It represents interactions in the form of messages from external systems/actors to BBS or from smart contracts to other smart contracts. The sequence model may refine architecture model and smart contract model [S1], [S3],[S23]. Bettin-Diaz et al [S18] suggest *data flow diagram* to identify how data is passed across running off-blockchain and on-blockchain and where it is processed.

As the deployed smart contracts cannot call external APIs, interfaces are implemented via front-end web technologies, such as JavaScript and HTML5, to enable on-blockchain components to interact with off-blockchain components. In this regard, an *interaction model* is useful for identifying important behavioral patterns of interactions and thus defining the integration points to be implemented for connecting on-blockchain and off-blockchain components [S28],[S34],[S47], [S56].

A blockchain-specific work-product during the design phase, as a result of *consensus mechanism design* task (as defined in section 5.2.2), is the *consensus model*. It shows how an agreement between all nodes in BBS is reached to accept a new block to the blockchain. This model reflects design decision parameters such as transaction throughput, latency, network bandwidth, rules, and incentives that should be enforced by a selected consensus mechanism.

Since software teams generate models for a purpose, the traceability and dependency among them, i.e., what models are predecessors/successors or master-subordinate to a certain model, in the whole BBS development process endeavor is intuitive. This is in line with the discussion of software requirements management and traceability. Gotel and Finkelstein [S58] explain that traceability is the ability to follow requirements in both forward and backward directions starting from their origins towards implementation, deployment, and maintenance phases. According to this line of argument, establishing the traceability among models in a BBS engineering endeavor enables better development process management, automatic forward and backward engineering, responsibility assignment to teams, and error detection. The review of the selected set of studies shows that traceability is weakly supported in the literature. Only a few works define the chain of models for the development process: [S6] (*smart contract template → smart contract model→ code model*), [S23] (*use-cases model* [main actors of BBS] → *class model* [BBS relationships with actors] → *sequence model* [realization of transactions by BBS], [S32] (*smart contract model → code model*, *deployment model→ configuration model*), and [S43] (*ontology models → smart contract model*).

5.3.2 Modeling languages

The notations and semantic rules imposed by a modelling language that is used in BBS engineering process offer threefold advantages (i) enabling precise expression of BBS aspects and generated work-products, (ii) providing a consistent way of communication among software teams and stakeholders, and (iii), increasing the automation level of code generation and test.

From the reviewed studies, it is found that Unified Modelling Language (UML) [64]– adopted by the Object Management Group (OMG) in 1997 as a de-facto for object-oriented modelling– is the most often adopted modeling language in BBS development process [S1],[S5],[S6],[S17],[S23],[S28],[S32],[S34],[S46],[S56]. An advantage of using UML class diagrams to represent the internal structure of a smart contract such as functions and data attributes is automatic smart contract code generation. Extending UML through stereotypes to cater necessary BBS-specific modeling requirements in BBS development is intuitive [S46]. The lack of capability for modeling different types of distributed ledger technologies has motivated Gorski and Bednarski [S32] to propose UML stereotypes and tagged values for generating deployment scripts and configuration files, organized as UML profile for *distributed ledger deployment*. Apart from UML, it can be seen that BPMN (Business Process Modeling Notation) [65], a de-facto for flow-oriented representation of core constructs of business processes, is used as a complementary to UML for business people. In this regard, a first promising attempt by Rocha and Ducasse [S53] suggests that using BPMN in BBS development process has advantages to identify (i) transactions, functions, and actors in collaborative business processes to map them to smart contracts and (ii) interactions –integration paint points– between off-blockchain and

on-blockchain components. For example, the *swimlane* notation which is the named box container can be used to specify interactions between legacy systems and BBS. Other alternatives notations to BPMN for the same purpose of business processes modeling, suggested by Almeida et al. [S51], are BDF (Block Diagram Flow) and BFPD (Block Flow Process Diagram).

Recall from Section 5.1, it tends to be seen that ontologies and related technologies including Web Ontology Language (WOL) and Semantic Web Rule Language (SWRL) are suitable for modeling smart contracts [S42],[S43]. In their work, Choudhury et al., [S43] provide a modeling framework for the automatic generation of smart contracts. The framework adopts ontologies to represent and encode business constraints and semantic rules available in unstructured business documents (legal contracts, tables, and charts). This standard representation via SWRL rules ensures accurate parsing and enables the automatic generation and test of smart contract templates in a given BBS application domain.

5.4 RQ4: What key roles are in a BBS development endeavor and what do they play?

The BBS development can be centered on the view of bringing people who collaborate to reach the final BBS frontier. They can be either producers in a software team who are responsible for creating, assessing, iterating, maintaining models/work-products or end-users who interact with BBS. BBS engineering relies on the availability of roles with technical expertise, business acuity, and well-defined skill portfolios that should be acquired and to ensure that they have a clear understanding of their roles. Development roles need to be tailored in accordance with the project settings. For example, the development of a BBS for public blockchain, as pointed out in Section 5.2.2, requires different BBS-specific roles with a smart contract development aiming at designing, testing, and deploying smart contracts to potentially unknown users of smart contracts participating at a peer-to-peer public network. Unlike a limited number of roles that are involved in BBS development for private blockchain type, more cooperation effort might be required for BBS development for a public blockchain. That is, broad arrangements between parties about the content and objectives of smart contracts should be made to ensure that preconditions are met before smart contracts are turned to executable codes.

Apart from the commonly-known roles in non-blockchain software engineering such as project manager/technical leaders developer, network administrator and data modeler, as equally emphasized in BBS [S1],[S13] there are a few new roles introduced in BBS engineering. As shown in Figure 8, they are broadly split into five distinct groups of (i) *core blockchain developer*, (ii) *blockchain software developer*, (iii) *systems integration engineer*, (iv) *legal professional*, and (v) *blockchain user* [S1],[S6],[S13],[S45],[S52],[S55],[S56]. Each role is associated with different responsibilities during the development lifecycle and all roles collaborate as a whole to accomplish the development of BBS. The role of a *core blockchain developer* refers to a development party who is responsible for designing blockchain platforms, APIs, protocols, network architecture, and security patterns related to blockchain technology. The *blockchain software developer* role, on the other hand, utilizes enabling foundations provided by the core blockchain developer to implement BBS running on blockchain platforms. Typically, a blockchain software developer is responsible for writing smart contracts that codify critical business logic in a secure way, implementing interactive front-end interface with BBS, backend systems, and maintaining the full stack of running BBS. Bosu et al. [S22] highlight three essential skills for blockchain software and core developers: (i) strong security programming background, (ii) networking knowledge for secure design of communication between distributed off-blockchain and on-blockchain components, (iii) cryptography mathematics for designing BBS algorithms and protocols. In conjunction with these skills, in-depth knowledge about scalable architecture design is an essential skill for the role of developers [S1],[S5]. *Systems integration engineers* play an important role. They are in charge of integrating all on-blockchain and off-blockchain components, assuring the satisfaction of interoperability requirements, seeking approvals of component changes, and ensuring the entire BBS functions correctly [S16].
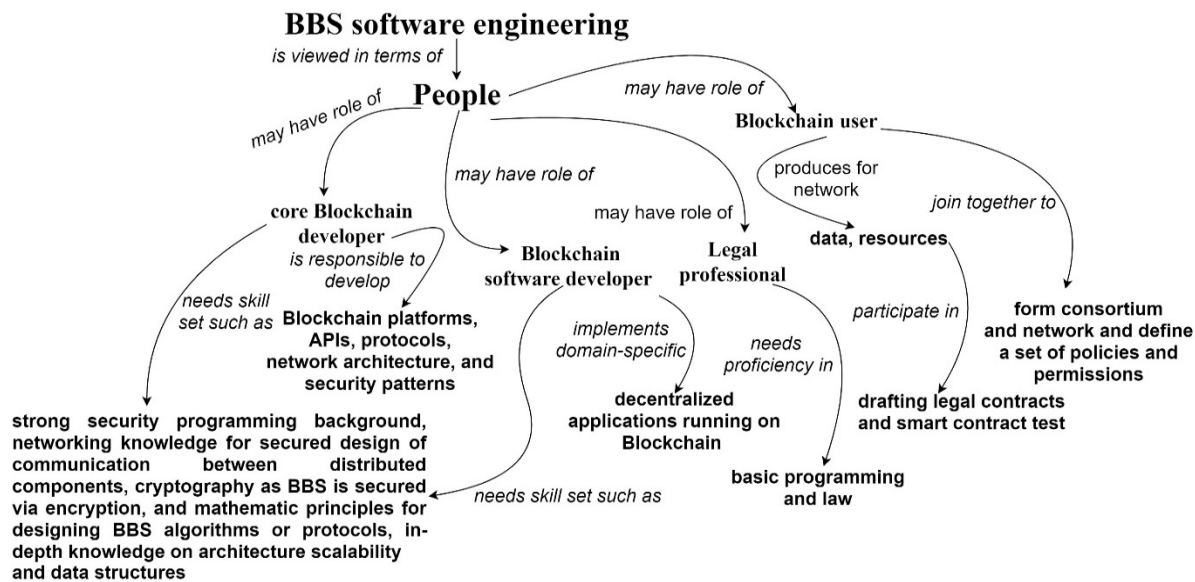
Fig. 8. *Role* conceptual map

Apart from the technical roles, BBS engineering also entails a portfolio of roles with expertise in finance and law domains. *Legal professionals* are needed between business-focused contractors and technical-focused software teams. This role is responsible for identifying potential contractors and partners. Giancaspro et al. [S46] and Almeida et al. [S51] highlight a need for a new generation of IT lawyer role in software teams for complex business transactions embarking on blockchain. The role that comes into play across different phases of the development process should have basic proficiency in programming and be responsible for drafting legal contracts and ensuring their clauses are adequately translated into relevant smart contract code. Rocha et al. [S53] point out that legal expertise is crucial to ensure test cases cover and verify the prose of a legal contract in smart contracts.

Finally, the role of *blockchain user* in a business model integrated with blockchain eco-system is to consume or produce data and contribute to resources that are beneficial to others [S2],[S12],[S17],[S31],[S34]. Blockchain users, join together to form a network and define a set of policies and permissions agreed by all as a consortium or private blockchain type architecture.

6 FUTURE RESEARCH DIRECTIONS

The review of the literature in Section 5 not only presents an overview of concepts and fundamental understanding of BBS engineering that is portrayed by our conceptual framework, but also reveals symptomatic of the needs for holistic and integrated engineering approaches in the future. The situation, literally, needs further research works to ameliorate the status quo of software engineering for BBS. The following research gaps, related to each aspect of the framework deserve further exploration.

6.1 Absence of process tailoring

It has been well-recognized that software engineering approaches should be tailored to the circumstances surrounding a project, the essence of which is truly stated by Sommerville and Ransom [66], p.93: *it is a truism that any method has to be adapted for the particular circumstances of use*. The view that Brook [67] calls as the *silver bullet*, i.e. one-size-fits-all assumption, is not a practical choice, continues to hold true and a persistent theme in BBS engineering. In recognition of this, Miraz and Ali [68] examine the suitability of adopting traditional software development lifecycles in BBS context and conclude that software teams may omit certain aspects in an engineering approach not from a position of ignorance, but from the pragmatic base that those aspects are not relevant to the functions of a selected blockchain platform, the choice of blockchain type, or project's setting itself. No matter how well-crafted, it might be impractical to find or design a single engineering approach applicable to all BBS projects and in totality as *imposed from above*. Software teams may have their in-house approach for software development,

however, it may confine its focus on some BBS development tasks and ignore others. For example, at the early phases, if a software team looks for recommendations on the feasibility of a BBS instance in a certain organization, the findings in [S24],[S46],[S47] are quite relevant to be considered by the team. On the other hand, if the team is rather in the design phase and seeks smart contract design techniques, then a piece of advice in the work by Frantz and Nowostawski [S57] for high-level design as well as work by [S31], [S46],[S47] for low-level design is applicable.

Given the specific situational factors and architectural decisions of BBS development context, as some listed in sections 5.2.1 and 5.2.2, one may logically conclude that BBS engineering approaches should be malleable or selected from a portfolio of approaches. A suggested solution in software engineering, which can be equally applied in BBS engineering, is *situational method engineering (SME)* [69]. SME is to tailor or create project-specific engineering methodologies via selecting appropriate method fragments from a method base and assembling them to construct a highly customized methodology. However, to the best of our knowledge, little research exists on process tailoring for BBS development. Questions like what key method fragments, e.g. process patterns (Section 5.1), are required to be populated in a method base, and how they can be assembled according to the project situation factors to tailor or create a custom-specific methodology have yet to be addressed in more depth in the future.

6.2 Requirements analysis is a dire problem

Whilst the topic of requirements analysis has achieved its maturity in conventional software engineering, it is currently less studied in BBS engineering. Apart from blockchain-specific characteristics, the requirements analysis for BBS does not significantly differ from any non-BBS. Recall from Section 5.2.1, software teams can still be referred to use the conventional requirements analysis techniques, as suggested by [S5],[S31],[S49]. On the contrary, the selected studies overlook questions like what requirements contribute *value* to BBS stakeholders. In other words, this is left as an unexplored area in BBS requirements analysis that what requirements constitute value and instigate a variety of stakeholders with diverging goals and commitment levels if they are addressed by smart contracts. This is consistent with the case study findings in Du et al., [S47] who accentuate the complexity of BBS for stakeholders and the fact that they primary care about if BBS being useful rather than what the underlying technology offers. As the definition of smart contracts and solution architecture depend on the elicitation of true requirements as well as legal contracts, a tighter connection between the analysis phase and the design phases is required to ensure smart contract errors are detected and fixed as soon as possible. More empirical studies are needed to get a better understanding of the criteria and techniques for selecting requirements when fulfilled by BBS will create added value for stakeholders.

6.3 Importance of model traceability

Some selected core papers (cf. Section 5.3) state that the creation of models has a vital role in BBS development to manage complexity, represent information at a different level of abstraction, and increase the automated model generation when the scale of transactions is large. However, few studies such as [S6],[S25],[S32],[S43] establishes a chain of models to be produced to reach final BBS. Leveraging well-established state of the art in MDE based software engineering, as pointed in Section 5.3.1, keeping predecessor-successor or master-subordinate traceability and relationships between models in a BBS engineering is important to enable (i) the evaluation of the degree to which legal contracts and security requirements are matched with executable smart contracts, (ii) the assessment of the degree to which each component in a BBS development justifies its existence, for example, whether or not a smart contract's function is traced to the prose of a legal contract that it satisfies, and (iii) automatic forward and backward model transformation of smart contracts. Future research can find embarking MDD based engineering approaches helpful in a seamless and automated transformation of intermediate BBS models that are an important avenue to reach the final BBS product.

6.4 Legacy system blockchain enablement is weakly supported

Legacy software systems operating and storing critical organizational data for years predate blockchain technology and the harness of distributed ledgers. It is important to realize that the term legacy system is often collocated with the old generation of technologies such as mainframes, monolithic architecture, file systems, communication protocols, and programming languages such as FORTRAN, COBOL, and C. A software system might have been developed using common technologies such as Microsoft .Net and J2EE, however, it may not support blockchain specific design considerations such as immutability, stateless/stateful smart contracts, off-blockchain/on-blockchain

components as discussed in Section 5.2.2. Such a system is subsumed under legacy if it is going to be moved to a new blockchain platform. Software re-engineering of these systems, allied to the large volume of organizational data, to make them blockchain-enabled is a crucial endeavor. This is due to reasons, for example, data migration, interoperability between blockchain and legacy platforms, mode of hosting, blockchain platform vendor lock-in, and security issues of transparency and openness to participant nodes in a blockchain network. According to these lines of argument, migrating large scale and complex legacy systems to blockchain platforms needs to be organized and anticipated in a systematic way.

Whether or not the above perception is valid, it is worth investigating how and what new practices a software team should incorporate into the development process to make legacy systems blockchain-enabled. A stream of research exists on the legacy system migration to SOA [29] and cloud platforms [9],[30]. This is timely to continue this research stream to the topic of migrating legacy systems to blockchain platforms. The presented framework in this survey, in light of the reported findings in [S38], is a starting point to set the scene for further explanation.

6.5   Need for stakeholder-driven engineering approaches

Presumably, the support of all stakeholders, amongst others, is a key aspect of any successful software engineering project. This is equally important in the context of BBS engineering project. Recall from Section 5.4, BBS engineering introduces more types of stakeholders as opposed to non-BBS engineering. These range from core blockchain developers to legal IT professionals, who may have multiple roles within the development lifecycle and have an interest in or are influenced by the lifecycle. In light of our general guidelines for BBS stakeholders presented in Section 5.4, a stakeholder-driven perspective for BBS engineering with an emphasis on roles and distributed teams/development is beneficial to the project management and governance through the entire development lifecycle. Researchers can also run empirical studies across multiple teams and organizations to identify *when* each stakeholder is involved, *what* roles are assigned to, *where* and *how* stakeholders influence and are influenced in the course of analysis, design, implementation and operation of BBS. The results of these studies can contribute to the identification of stakeholder-driven BBS engineering approaches that are significant in the areas of (i) BBS requirements analysis and design phase, (ii) BBS adoption in organizations, and (iii) project performance measurements.

7   THREATS TO VALIDITY

We divide the limitations of our survey in terms of *internal* and *external* validity [70]. The former bears on factors that we might have been unaware of or unable to control them whilst the latter concerns with threats that negatively affect the generalizability of our findings.

As far as the internal validity is concerned, due to a sheer volume of published works in academic and multi-vocal literature on blockchain, it was infeasible to conduct an exhaustive literature search to identify all works in the digital libraries related to the research questions stated in our work. One reason for this issue is that a key inclusion criterion in conducting our SLR (Section 4.1). We have given the priority to identify well-cited papers with a proper validation in a real-world application case from the literature in order to increase the reliability of the presented conceptual framework and reported findings. We have been reluctant to consider opinion papers, technical reports, white papers, and books in our survey. Moreover, we observed that some important papers that had not directly discussed the software engineering of BBS, nevertheless, their identified challenges and propositions were truly related to the focus and scope of this survey. The blockchain field is still an immature area and its literature is overwhelmed with interpretations and variants of concepts that might not necessarily be homogeneous. We found that some papers do not use the term of software engineering in any section, but, they were important in crafting our framework and sharing important challenges and proposed countermeasures related to the subject of this survey. For instance, any of our search queries combining the key terms like *software engineering* and *blockchain* over the digital library directly results in identifying papers like [S26], [S30], as referred to in Section 5.1 and classified under the design phase (Section 5.2.2). Inversely, their inclusion in the selected papers was a result of several rounds of the snowballing technique. Unavoidably, the choice of the papers might have been influenced by subjective bias. To lessen this issue, two co-authors conducted the literature review, paper selection, content analysis, and framework generation whilst senior researchers oversaw the whole survey. We conducted the purposeful snowballing technique and the careful scanning of the identified studies from the literature to select the ones that strictly satisfied the selection criteria

(Section 4.1). An observation in the selected studies reveals that the highest number of the studies are affiliated with Australia as same as the authors' affiliation. The reason is that we started with an initial set of papers from our domestic academic colleagues and expanded them to identify new papers through reviewing the related studies and references that were cited in this set

As for the external validity, we do not claim to provide complete coverage and inclusivity of the presented conceptual framework. A less discussed aspect in Section 5.2.3, yet important, in the presented conceptual framework, is the aspect of the *tool* support for BBS. It is associated with the aspects of *process* and *modeling* in the framework. As our survey intends to provide an overarching layer for BBS development by abstracting pure technical and dispersed literature, we believed that the aspect of *tool*, despite the availability of some dedicated BBS tools (Section 5.2.3), can be either a discussion on the way of operationalization of the framework's fragments or a topic of future surveys based on our proposed framework. Finally, whilst the derivation of the framework through Grounded Theory has been iterative and with several refinements in the literature source and coding procedures, we don't claim about the generalizability of the presented framework and accounted findings beyond 58 source papers in the current survey.

8 CONCLUSION

We presented a systematic literature review on software engineering for BBS. We addressed the most relevant and up-to-date survey questions synthesizing published material available in this area. We explored the current state-of-the literature and identified important future research directions in light of the fundamental aspects of the conceptual software engineering framework for BBS. In terms of the process aspect, firstly, the discussion on requirements analysis has received much less attention from researchers. Further research can investigate how or which added-value requirements should be selected to realize by BBS. Secondly, some researchers have proposed situational characteristics upon which BBS development process should be tailored. Whereas the need for a flexible and bespoke BBS development process is acknowledged, there is a dearth of research dealing with this issue and little is known about how tailoring is applied for BBS engineering projects. Thirdly, considering the importance of legacy software systems supporting back-end functions of IT-based organizations and storing critical data, the minority of the identified papers discuss legacy software re-engineering to blockchain, for example, steps to identify and integrate core business logic, e.g. Web-Service, to smart contracts. Migrating legacy software to blockchain has recently received attention. A future trend may particularly focus on software re-engineering for making legacy systems blockchain-enabled. Fourthly, in terms of the modeling aspect, several models are generated during a BBS development lifecycle ranging from requirements to smart contract design elements, and down to smart contract code fragments. Managing the chain of these models as a means to understand logical traceability and bounded dependency among these models plays an important role to enable automated BBS design in view of requirements. Techniques related to the traceability of models in BBS development are partially captured in the literature, which stimulates another possible future research. Fifthly, from the aspect of stakeholders, bearing in mind that BBS development involves more types of technical and non-technical stakeholders than many conventional software systems, with diverse goals in using BBS and different level of engagement, stewardship, cooperation, and incentives, this survey provides input for future research with a focus on stakeholder-driven BBS development.

The eventual upshot is that software engineering for BBS is not as straightforward as for many non-BBS ones. Contaminated with partiality and subjectivity, adopting an ad-hoc approach is hardly safe and secure for business-critical BBS. More research is necessary to move from ad-hoc BBS development to more disciplined BBS engineering and to enhance the maturity of this research field. To pave this path, we explored the current research state of the literature and identified important future research directions in light of the presented conceptual framework. We hope that researchers benefit from highlighted ideas that are shared in this survey as entry points to tackle unaddressed issues. The practitioners can incorporate the findings in this survey into their in-house software development approach to enhance its capability for BBS development.
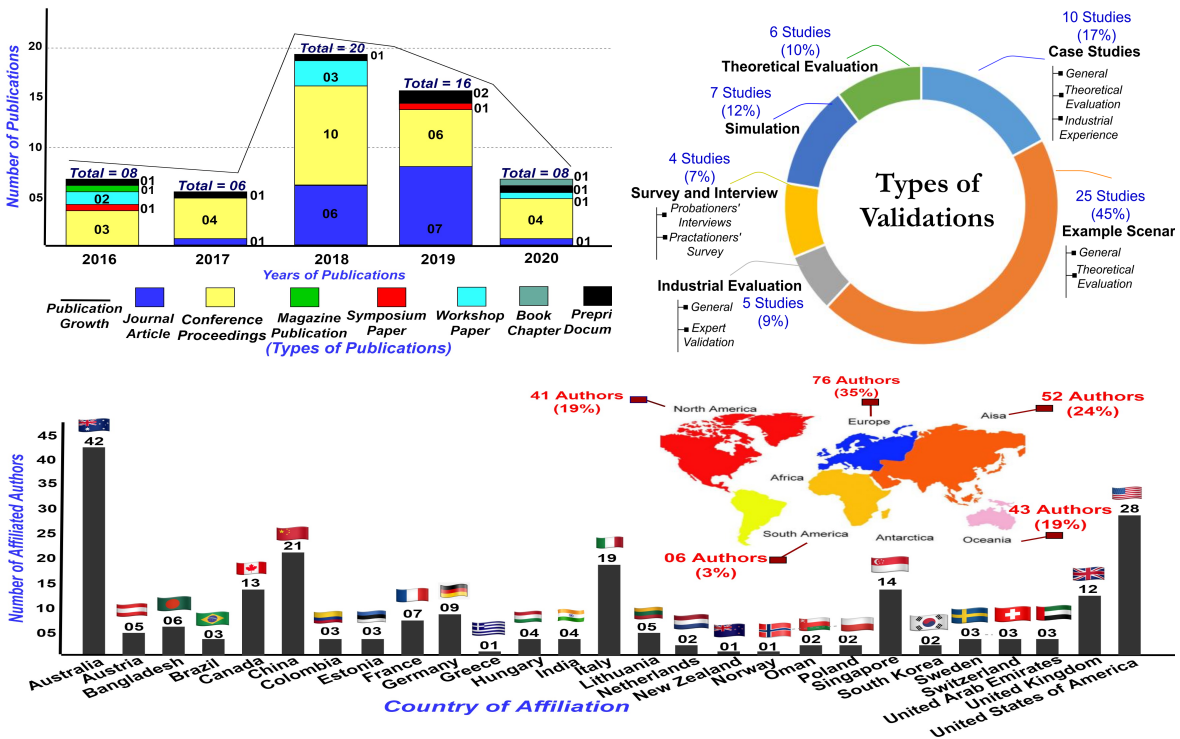
Acknowledgements

Appendix A: Selected core papers

#	Author and Title	Channel	Source	Year	Validation Technique
[S1]	C. Lin, D. He, X. Huang, *BSeIn: A blockchain-based secure mutual authentication with fine-grained access control system for industry*	J	Elsevier	2018	Simulation
[S2]	L. Hang, D.H. Kim, *Design and implementation of an integrated IoT blockchain platform for sensing data integrity*	J	Sensors	2019	Simulation
[S3]	M. Marchesi, L. Marchesi, R. Tonelli, *An agile software engineering method to design blockchain applications*	C	ACM	2018	Example scenario
[S4]	I. Karamitsos, M. Papadaki, N.B. Al Barghuthi, *Design of the blockchain smart contract: A use case for real estate*	J	Scientific Research Publishing	2018	Example scenario
[S5]	M. Jurgelaitis, V. Drungilas, L. Ceponiene, R. Butkiene, E. Vaiciukynas, *Modelling principles for blockchain-based implementation of business or scientific processes*	C	IVUS	2019	Example scenario
[S6]	Y. Shi, Z. Lu, R. Tao, Y. Liu, Z. Zhang, *A trading model based on legal contracts using smart contract templates*	C	Springer	2019	Example scenario, Theoretical evaluation
[S7]	Badr, A., Rafferty, L., Mahmoud, Q.H., Elgazzar, K., Hung, P.C., *A permissioned blockchain -based system for verification of academic records*	C	IEEE	2019	Simulation
[S8]	G. Destefanis, M. Marchesi, M. Ortu, *Smart contracts vulnerabilities: a call for blockchain software engineering?*	W	IEEE	2018	Case study
[S9]	P. Chakraborty, R. Shahriyar, A. Iqbal, A. Bosu, *Understanding the software development practices of blockchain projects: a survey*	C	ACM	2018	Survey of practitioners
[S10]	C. Udokwu, H. Anyanka, A. Norta, *Evaluation of Approaches for Designing and Developing Decentralized Applications*	C	ACM	2020	Case study, Theoretical evaluation
[S11]	Y. Yuan, F.Y. Wang, *Towards blockchain -based intelligent transportation systems*	C	IEEE	2016	Example scenario
[S12]	W. Dai, C. Wang, C. Cui, H. Jin, X. Lv, *blockchain-Based Smart Contract Access Control System*	C	IEEE	2019	Simulation
[S13]	W. Zou, D. Lo, P.S. Kochhar, X. D. Le, X. Xia, Y. Feng, Z. Chen, B. Xu, *Smart contract development: Challenges and opportunities*	J	IEEE	2019	Survey and semi-structured interview of practitioners
[S14]	S. Lilani, D. Malani, J. Modi, F. Soni, *Securing the Software Development Life Cycle (SDLC) with a blockchain Oriented Development Approach*	C	arXiv	2019	Case study
[S15]	B. Marino and A. Juels, *Setting standards for altering and undoing smart contracts*	S	Springer	2016	Example scenario
[S16]	Yue, K., *Blockchain-augmented organizations*	C	AIS Library	2020	Example scenario
[S17]	L. Luu, D.-H. Chu, H. Olickel, *Making smart contracts smarter*	C	ACM	2016	Simulation
[S18]	R. Bettín-Díaz, A.E. Rojas, *Methodological approach to the definition of a blockchain system for the food industry supply chain traceability*	C	Springer	2018	Case study, industrial experience
[S19]	X. Xu, Q. Lu, Y. Liu, L. Zhu, H. Yao, A.V. Vasilakos, *Designing blockchain -based applications a case study for imported product traceability*	J	Elsevier	2019	Case study
[S20]	W. Cai, Z. Wang, J.B. Ernst, Z. Hong, C. Feng, V.C. Leung, *Decentralized applications: The blockchain -empowered software system*	J	IEEE	2018	Industrial experience
[S21]	N.B. Truong, K. Sun, G.M. Lee, Y. Guo, *Gdpr-compliant personal data management: A blockchain-based solution*	J	IEEE	2019	Example scenario
[S22]	A. Bosu, A. Iqbal, R. Shahriyar, P. Chakraborty, *Understanding the motivations, challenges and needs of blockchain software developers: a survey*	J	Springer	2019	Survey of practitioners
[S23]	L. Marchesi, M. Marchesi, R. Tonelli, *ABCDE--Agile Block Chain Dapp Engineering*	P	Elsevier	2019	Example scenario
[S24]	B. A. Scriber, *A framework for determining blockchain applicability*	J	IEEE	2018	Semi-structured interviews
[S25]	L. Cocco, A. Pinna, G. Meloni, *A blockchain Oriented Software Application in the Revised Payments Service Directive context*	W	IEEE/ACM	2020	Example scenario
[S26]	X. Xu, C. Pautasso, L. Zhu, Q. Lu, I. Weber, *A pattern collection for blockchain-based applications*	C	ACM	2018	Theoretical evaluation
[S27]	G. Al-Mazrouai, S. Sudevan, *Managing blockchain Projects with Agile Methodology*	C	Springer	2020	Example scenario
[S28]	N. Aldred, L. Baal, G. Broda, S. Trumble, Q. H. Mahmoud, *Design and Implementation of a blockchain-based Consent Management System*	P	arXiv	2019	Example scenario, theoretical evaluation
[S29]	Q. Lu, A. Tran, I. Weber, *Integrated Model-Driven Engineering of blockchain Applications for Business Processes and Asset Management*	P	arXiv	2020	Example scenario
[S30]	I. Weber, X. Xu, R. Riveret, G. Governatori, *Untrusted business process monitoring and execution using blockchain*	P	Springer	2016	Simulation
[S31]	P. Garamvölgyi, I. Kocsis, B. Gehl, A. Klenik, *Towards Model-Driven Engineering of Smart Contracts for Cyber-Physical Systems*	C	IEEE/IFIP	2018	Example scenario

[S32]	T. Górski, J. Bednarski, *Applying Model-Driven Engineering to Distributed Ledger Deployment*	J	IEEEE	2020	Example scenario
[S33]	F. Glaser, *Pervasive decentralization of digital infrastructures: a framework for blockchain enabled system and use case analysis*	C	IEEE	2017	Theoretical evaluation
[S34]	A. Mavridou, A. Laszka, E. Stachtiari, A. Dubey, *VeriSolid: Correct-by-design smart contracts for Ethereum*	C	IEEE	2019	Example scenario
[S35]	P. Zhang, J. White, D. C. Schmidt, G. Lenz, *Applying software patterns to address interoperability in blockchain-based healthcare apps*	P	arXiv	2017	Case study
[S36]	M. Wöhrer, U. Zdun, *Design patterns for smart contracts in the Ethereum ecosystem*	C	IEEE	2018	Simulation
[S37]	Y. Liu, Q. Lu, X. Xu, L., *Applying design patterns in smart contracts*	C	Springer	2018	Case study, Industrial experience
[S38]	H. D. Bandara, X. Xu, I. Weber, *Patterns for blockchain Data Migration, in Proceedings of the European Conference on Pattern Languages of Programs.*	C	ACM	2020	Theoretical evaluation
[S39]	M. Wohrer, U. Zdun, *Smart contracts: security patterns in the Ethereum ecosystem and solidity*	W	IEEE	2018	Theoretical evaluation, Example scenario
[S40]	M. Bartoletti, L. Pompianu, *An empirical analysis of smart contracts: platforms, applications, and design patterns*	C	Springer	2017	Case studies
[S41]	J. De Kruijff , H. Weigand, *Understanding the blockchain using enterprise ontology*	C	Springer	2017	Theoretical evaluation
[S42]	H. M. Kim, M. Laskowski,, Management, *Toward an ontology-driven blockchain design for supply-chain provenance*	J	Wiley	2018	Example scenario
[S43]	O. Choudhury, N. Rudolph, I. Sylla, N. Fairoza,, A. Das, *Auto-generation of smart contracts from domain-specific ontologies and semantic rules*	C	IEEE	2018	Case study
[S44]	H. Baqa, N. B. Truong, N. Crespi, G. M. Lee, F. Le Gall, *Semantic Smart Contracts for blockchain-based Services in the Internet of Things*	S	IEEE	2019	Example scenario
[S45]	G. Governatori, F. Idelberger, Z. Milosevic, *On legal contracts, imperative and declarative smart contracts, and blockchain systems*	J	Springer	2018	Example scenario, Theoretical evaluation
[S46]	M. Giancaspro, *Is a smart contract 'really a smart idea? Insights from a legal perspective*	J	Elsevier	2017	Expert validation, industrial experience
[S47]	W. D. Du, S. L. Pan, D. E. Leidner, *Affordances, experimentation and actualization of FinTech: A blockchain implementation study*	J	Elsevier	2019	Interview
[S48]	A. M. Langer, *Blockchain analysis and design*	B	Springer	2020	Example scenario
[S49]	C. Hebert, F. Di Cerbo, *Secure blockchain in the enterprise: A methodology*	j	Elsevier	2019	Example scenario
[S50]	J. Plansky, T. O'Donnell, K. Richards, *A strategist's guide to blockchain*	M	PWC	2016	Industrial experience
[S51]	S. Almeida, A. Albuquerque, A. Silva, *An approach to develop software that uses blockchain*	C	Springer	2018	Example scenario, Theoretical evaluation
[S52]	G. Fridgen, J. Lockl, *A solution in search of a problem: a method for the development of blockchain use cases*	C	AIS e-Lib	2018	Case study
[S53]	H. Rocha, S. Ducasse, *Preliminary steps towards modeling blockchain oriented software*	W	IEEE	2018	Example scenario
[S54]	X. Xu, C. Pautasso, L. Zhu, V. Gramoli, A. Ponomarev, S. Chen, *The blockchain as a software connector*	W	IEEE	2016	Industrial experience
[S55]	X. Xu, I. Weber, M. Staples, *A taxonomy of blockchain-based systems for architecture design*	C	IEEE	2017	Theoretical evaluation
[S56]	T. Hardjono, A. Lipton, A. Pentland, *Towards a design philosophy for interoperable blockchain systems*	P	arXiv	2018	Example scenario
[S57]	C. K. Frantz, M. Nowostawski, *From institutions to code: Towards automated generation of smart contracts*	W	IEEE	2016	Example scenario
[S58]	C. D. Clack, V. A. Bakshi, L. Braine, *Smart contract templates: foundations, design landscape and research directions*	P	arXiv	2016	Example scenario

Appendix B: Demographic information of core papers

REFERENCES

[1]   "Deloitte's 2019 globalc Blockchain survey-Blockchain gets down to business," 2019: Available at: https://www2.deloitte.com/content/dam/Deloitte/se/Documents/risk/DI_2019-global-blockchain-survey.pdf (last access October 2020).

[2]   A. Bosu, A. Iqbal, R. Shahriyar, and P. Chakraborty, "Understanding the motivations, challenges and needs of Blockchain software developers: a survey," *Empirical Software Engineering journal,* vol. 24, no. 4, pp. 2636-2673, 2019.

[3]   S. Porru, A. Pinna, M. Marchesi, and R. Tonelli, "Blockchain-oriented software engineering: challenges and new directions," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017: IEEE, pp. 169-171.

[4]   K. Costello, "Gartner report," Jun 2019: Available at: https://www.gartner.com/en/newsroom/press-releases/2019-07-03-gartner-predicts-90--of-current-enterprise-blockchain (last access Feb 2020).

[5]   Gartner, "Gartner Predicts," *Available at: https://www.gartner.com/en/newsroom/press-releases/2019-07-03-gartner-predicts-90--of-current-enterprise-blockchain (last accessed:October 2021)*.

[6]   M. Avital, R. Beck, J. King, M. Rossi, and R. Teigland, "Jumping on the Blockchain Bandwagon: Lessons of the Past and Outlook to the Future," *in Proceedings of the 37th International Conference on Information Systems, B. Fitzgerald and J. Mooney (eds.), Dublin, Ireland,* 2016.

[7]   M. Risius and K. Spohrer, "A blockchain research framework," *Journal of Business Information Systems Engineering,* vol. 59, no. 6, pp. 385-409, 2017.

[8]   P. Chakraborty, R. Shahriyar, A. Iqbal, and A. Bosu, "Understanding the software development practices of blockchain projects: a survey," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2018, pp. 1-10.

[9]   M. Fahmideh, F. Daneshgar, G. Low, and G. Beydoun, "Cloud migration process—A survey, evaluation framework, and open challenges," *Journal of Systems and Software,* vol. 120, pp. 31-69, 2016.

[10]   M. Fahmideh and D. Zowghi, "An exploration of IoT platform development," *Information Systems,* vol. 87, p. 101409, 2020.

[11]   L. Marchesi, M. Marchesi, and R. J. a. p. a. Tonelli, "ABCDE--Agile Block Chain Dapp Engineering," 2019.

[12]   M. Swan, *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015.

[13]   M. Pilkington, "11 Blockchain technology: principles and applications," *Research handbook on digital transformations,* p. 225, 2016.

[14]   J. Yli-Huumo, D. Ko, S. Choi, S. Park, and K. Smolander, "Where is current research on blockchain technology?—a systematic review," *PloS one, 11(10), e0163477,* vol. 11, no. 10, p. e0163477, 2016.

[15]   J. Liu and Z. Liu, "A survey on security verification of blockchain smart contracts," *IEEE Access,* vol. 7, pp. 77894-77904, 2019.

[16]   B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering – A systematic literature review," *Information and software technology,* vol. 51, no. 1, pp. 7-15, 2009, doi: http://dx.doi.org/10.1016/j.infsof.2008.09.009.

[17]   B. G. Glaser and A. L. Strauss, *The discovery of grounded theory: Strategies for qualitative research*. Transaction Books, 2009.

[18]   M. Crosby, P. Pattanayak, S. Verma, and V. Kalyanaraman, "Blockchain technology: Beyond bitcoin," *Applied Innovation,* vol. 2, pp. 6-10, 2016.

[19]   Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *International Journal of Web and Grid Services,* vol. 14, no. 4, pp. 352-375, 2018.

[20]   F. Casino, T. K. Dasaklis, and C. Patsakis, "A systematic literature review of blockchain-based applications: current status, classification and open issues," *Telematics and Informatics 36 (2019): 55-81,* vol. 36, pp. 55-81, 2019.

[21]   N. Szabo, "Formalizing and securing relationships on public networks," *First monday,* 1997.

[22]   C. D. Clack, V. A. Bakshi, and L. Braine, "Smart contract templates: foundations, design landscape and research directions," *https://arxiv.org/abs/1608.00771,* 2016.

[23]   I. Sommerville, "Software Engineering," *7th ed. Addison-Wesley, Reading, Mass,* 2004.

[24]   D. H. Ingalls, "The Smalltalk-76 programming system design and implementation," in *Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, 1978, pp. 9-16.

[25]   S. L. Pfleeger, "Albert Einstein and empirical software engineering," *Computer,* vol. 32, no. 10, pp. 32-38, 1999.

[26]   G. Cugola, C. Ghezzi, and Practice, "Software Processes: a Retrospective and a Path to the Future," *Software Process: Improvement,* vol. 4, no. 3, pp. 101-123, 1998.

[27]   A. Fuggetta, "Software process: a roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, 2000, pp. 25-34.

[28]   S. Lane and I. Richardson, "Process models for service-based applications: A systematic literature review," *Journal Information and Software Technology,* vol. 53, no. 5, pp. 424-439, 2011.

[29]   M. Razavian and P. Lago, "A systematic literature review on SOA migration," *Journal of Software: Evolution and Process,* vol. 27, no. 5, pp. 337-372, 2015.

[30]   M. Fahmideh, F. Daneshgar, and F. Rabhi, "Cloud migration: methodologies: preliminary findings," in *European Conference on Service-Oriented and Cloud Computing–CloudWays 2016 Workshop*, 2016d.

[31]   A. Davoudian and M. Liu, "Big Data Systems: A Software Engineering Perspective," *ACM Computing Surveys,* vol. 53, no. 5, pp. 1-39, 2020.

[32]   A. Meidan, J. A. García-García, I. Ramos, and M. J. Escalona, "Measuring software process: a systematic mapping study," *Journal of ACM Computing Surveys,* vol. 51, no. 3, pp. 1-32, 2018.

[33]   R. Ramsin and R. F. Paige, "Process-centered review of object oriented software development methodologies," *ACM Computing Surveys (CSUR),* vol. 40, no. 1, p. 3, 2008.

[34]   A. Panarello, N. Tapas, G. Merlino, F. Longo, and A. Puliafito, "Blockchain and iot integration: A systematic survey," *Sensors,* vol. 18, no. 8, p. 2575, 2018.

[35]   X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, "A survey on the security of blockchain systems," *Future Generation Computer Systems,* vol. 107, pp. 841-853, 2020.

[36]   B. K. Mohanta, D. Jena, S. S. Panda, and S. Sobhanayak, "Blockchain technology: A survey on applications and security privacy challenges," *Journal Internet of Things,* vol. 8, p. 100107, 2019.

[37]   I. Konstantinidis, G. Siaminos, C. Timplalexis, P. Zervas, V. Peristeras, and S. Decker, "Blockchain for business applications: A systematic literature review," in *International Conference on Business Information Systems*, 2018: Springer, pp. 384-399.

[38]   A. Vacca, A. Di Sorbo, C. A. Visaggio, and G. Canfora, "A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges," *Journal of Systems and Software,* vol. 174, p. 110891, 2021.

[39]   V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative research in psychology,* vol. 3, no. 2, pp. 77-101, 2006.

[40]   C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, 2014: ACM, p. 38.

[41]   M. B. Miles and A. M. Huberman, *Qualitative data analysis: An expanded sourcebook*. Sage, 1994.

[42]   K.-J. Stol, P. Ralph, and B. Fitzgerald, "Grounded theory in software engineering research: a critical review and guidelines," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 120-131.

[43]   A. L. Strauss and J. Corbin, *Basics of Qualitative Research: Grounded Theory Procedure and Techniques*. Sage publications Newbury Park, CA, 1990.

[44]   R. S. Pressman, *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005.

[45]   D. Avison and G. Fitzgerald, *Information systems development: methodologies, techniques and tools*. McGraw Hill, 2003.

[46]   M. Fahmideh, F. Daneshgar, F. Rabhi, and G. Beydon, "A generic cloud migration process model," *European Journal of Information Systems,* 2018.

[47]   M. F. Gholami, J. Habibi, F. Shams, and S. Khoshnevis, "Criteria-Based evaluation framework for service-oriented methodologies," in *Computer Modelling and Simulation (UKSim), 2010 12th International Conference on*, 2010: IEEE, pp. 122-130.

[48]   S. Lane and I. Richardson, "Process models for service-based applications: A systematic literature review," *Information and Software Technology,* vol. 53, no. 5, pp. 424-439, 2011.

[49]   C. Gonzalez-Perez and B. Henderson-Sellers, *Metamodelling for software engineering*. Wiley Publishing, 2008.

[50]   I. I. 2007, "Software Engineering -- Metamodel for Development Methodologies ISO/IEC 24744:2007," *ISO/IEC Press,* 2007.

[51]   K. Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, "The agile manifesto," ed: February, 2001.

[52]   L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, "A survey of DevOps concepts and challenges," *ACM Computing Surveys (CSUR),* vol. 52, no. 6, pp. 1-35, 2019.

[53]   V. Lenarduzzi, M. I. Lunesu, M. Marchesi, and R. Tonelli, "Blockchain applications for Agile methodologies," in *Proceedings of the 19th International Conference on Agile Software Development: Companion*, 2018, pp. 1-3.

[54]   S. Beydeda, M. Book, and V. Gruhn, *Model-driven software development*. Springer, 2005.

[55]   C. Atkinson and T. Kuhne, "Model-driven development: a metamodeling foundation," *Software, IEEE,* vol. 20, no. 5, pp. 36-41, 2003.

[56]   A. G. Kleppe, J. Warmer, W. Bast, and M. Explained, "The model driven architecture: practice and promise," ed: Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2003.

[57]   D. S. Frankel, *Model driven architecture applying MDA*. John Wiley & Sons, 2003.

[58]   P. B. Kruchten, "The 4+ 1 view model of architecture," *IEEE software,* vol. 12, no. 6, pp. 42-50, 1995.

[59]   L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley Professional, 2003.

[60]   J. O. Coplien and D. C. Schmidt, *Pattern languages of program design*. ACM Press/Addison-Wesley Publishing Co., 1995.

[61]   N. F. Noy and D. L. McGuinness, "Ontology development 101: A guide to creating your first ontology," ed: Stanford knowledge systems laboratory technical report KSL-01-05 and …, 2001.

[62]   A. De Nicola, M. Missikoff, and R. Navigli, "A software engineering approach to ontology building," *Information systems,* vol. 34, no. 2, pp. 258-275, 2009.

[63]   E. Androulaki *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1-15.

[64]   OMG, "Object Management Group (OMG). 'Unified modeling language specification (v1.5)," 2003.

[65]   O. Omg, R. Parida, and S. Mahapatra, "Business process model and notation (bpmn) version 2.0," *Object Management Group,* vol. 1, no. 4, 2011.

[66]   I. Sommerville and J. Ransom, "An empirical study of industrial requirements engineering process assessment and improvement," *ACM Transactions on Software Engineering and Methodology (TOSEM),* vol. 14, no. 1, pp. 85-117, 2005.

[67]   F. Brooks and H. Kugler, *No silver bullet*. April, 1987.
[68]   M. H. Miraz and M. Ali, "Blockchain Enabled Smart Contract Based Applications: Deficiencies with the Software Development Life Cycle Models," *arXiv preprint arXiv:2001.10589,* 2020.
[69]   A. F. Harmsen, J. Brinkkemper, and J. H. Oei, *Situational method engineering for information system project approaches*. University of Twente, Department of Computer Science, 1994.
[70]   C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.