Open Source Blockchain Software Health: A Theoretical Framework

Jeff Nijsse

A thesis submitted to

Auckland University of Technology

in fulfilment of the requirements for the degree of

Doctor of Philosophy

School of Engineering, Computer, and Mathematical Sciences

2024

Abstract

The thesis combines research into blockchain software with the health of software ecosystems to produce a theoretical framework for open source blockchain health. The motivation comes from the blockchain trilemma, which argues that a blockchain network can achieve only two out of the three design objectives from decentralisation, security, and scalability. Between decentralisation and scalability, the objective is to investigate blockchain software from a health perspective to determine areas for improvement.

To fulfil these objectives, the study formulates a series of three focused research questions. The first is derived from the blockchain trilemma and asks: What are the factors that influence blockchain consensus? The second looks at software systems as a whole and asks: What is a definition of software health? The final research question unites software health and open source software (OSS) blockchain projects asking: What is included in a comprehensive model of OSS blockchain health?

To help answer these questions, hypotheses are formed that can then be tested within the overall methodology of Design Science Research (DSR). DSR allows for an iterative process where findings help inform processes and is particularly well suited for research in socio-technical fields. The focus of DSR is on the output artefacts that can be disseminated for further assessment and contribution to the field.

Among its contributions, the study offers a re-characterisation of the blockchain trilemma, distilling it into a dilemma that navigates between consensus methods, which are tied to decentralisation, and performance factors correlated with scaling capabilities. This, through the first question leads to the artefact of a taxonomy of blockchain consensus methods that reveals the landscape of algorithms underpinning decentralised networks.

In addition, the study introduces a conceptualisation of software health, drawn from literature in natural, business, and software ecosystems. It posits that the health of a software ecosystem is a composite construct comprising sustainability, robustness, and niche occupation. Sustainability is further decomposed into interest and engagement.

To operationalise the definition of health the study employs exploratory factor analysis

to search for latent constructs from specific metrics identified in the literature. General Interest is gauged through the observed variables of forks, stars, and mentions, while Developer Engagement consists of commits, pull requests, comments, and contributors. Software Robustness is measured using the metrics criticality, time since the last update, geographic distribution of contributors, and market capitalisation ranking. These metrics are empirically substantiated through confirmatory factor analyses.

Structural equation modelling is used to add a structural element to the latent factors. The study follows a framework for developing theory in Information Systems to derive a theoretical framework for software health. The framework is the prime artefact of the work, not only advancing scholarly discourse and contributing to knowledge, but also yielding actionable guidance applicable to a variety of stakeholders, ranging from project managers to volunteer open source developers.

In summary, the study contributes to both academic and practical realms by providing a methodologically rigorous, empirically substantiated framework, and metrics for assessing the health of blockchain projects in the OSS ecosystem.

To Satoshi,
Your clear vision
and selfless contribution
changed everything.

Contents

List of Figures

List of Tables

List of Acronyms

2PC	Two-Phase Commit
3PC	Three-Phase Commit
ACM	Association for Computing Machinery
ADR	Action Design Research
AIS	Association for Information Systems
AMI	Average Mutual Information
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
ATM	Automatic Teller Machine
BFT	Byzantine Fault Tolerance
BIC	Bayesian Information Criterion
BN	Bayesian Network
BNPA	Bayesian Network Path Analysis
BSD	Berkeley Software Distribution
CABRA	Comprehensive Academic Bitcoin Research Archive
CAP	Consistency, Availability, and Partition tolerance
CC0	Creative Commons Zero
CB-SEM	Covariance-Based SEM
CFA	Confirmatory Factor Analysis
CFI	Comparative Fit Index
CMC	CoinMarketCap
CPL	Common Public License
DAG	Directed Acyclic Graph
DAO	Decentralized Autonomous Organization
DoF	Degrees of Freedom
DPoS	Delegated Proof-of-Stake
DSR	Design Science Research
DSRM	Design Science Research Methodology
ECC	Elliptic Curve Cryptography
EFA	Exploratory Factor Analysis
EVM	Ethereum Virtual Machine
FBA	Federated Byzantine Agreement
FEDS	Framework for Evaluation in Design Science
FLOSS	Free/Libre Open Source Software
FOSS	Free Open Source Software

FSF	Free Software Foundation
GPL	GNU Public License
ICO	Initial Coin Offering
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IS	Information Systems
KMO	Kaiser-Meyer-Olkin
LoC	Lines of Code
ML	Maximum Likelihood
NPL	Netscape Public License
OSI	Open Source Initiative
OSSF	Open Source Software Foundation
p2p	Peer-to-Peer
PBFT	Practical Byzantine Fault Tolerance
PLS-SEM	Partial Least Squares SEM
PoET	Proof of Elapsed Time
PoH	Proof-of-History
PoS	Proof-of-Stake
PoW	Proof-of-Work
PoX	Proof-of-X
PR	Pull Request
RMSE	Root Mean Square Error
RMSEA	Root Mean Square Error of Approximation
SECO	Software Ecosystem
SEM	Structural Equation Modeling
SHA	Secure Hash Algorithm
SHARD	System for Highly Available Replicated Data
SMaRt	BFT State Machine Replication
SQL	Structured Query Language
SRMR	Standardized Root Mean Residual
SS	Sum of Squared loadings
TLI	Tucker-Lewis Index
TLS	Transport Layer Security
TPS	Transactions Per Second
ZK	Zero-Knowledge
ZKP	Zero-Knowledge Proof

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.

Jeff Nijsse

Publications

Nijsse, J., & Litchfield, A. (2023). Towards a structural equation model of open source blockchain software health. arXiv. https://arxiv.org/abs/2310.20277

Nijsse, J., & Litchfield, A. (2023). Identifying Developer Engagement in Open Source Software Blockchain Projects through Factor Analysis. *56th Hawaii International Conference on System Sciences*, 5333-5342. https://hdl.handle.net/10125/103285

Nijsse, J. (2022). Mining GitHub to identify open-source software health in blockchain projects. *Rangahau Aranga: AUT Graduate Review*, *1*(1). https://doi.org/10.24135/rangahau-aranga.v1i1.84

Nijsse, J., & Litchfield, A. (2020). A taxonomy of blockchain consensus methods. *Cryptography*, *4*(4). https://doi.org/10.3390/cryptography4040032

Acknowledgements

Few things happen in a vacuum, and thus there are a number of folks that come to mind that deserve their name in print in recognition of the parts they have played along the way.

From my alma matter, Windsor, Dr. Bruce for vehicle dynamics and supporting our SAE Aerodesign team, and writing letters of recommendation, thank you for being accessible and supportive; Dr. Fartaj for always asking pseudo-random students questions, thank you for convincing me I knew something about thermodynamics.

From the the Institute at Dufferin & Steeles, Pradeep Jha for all your assistance; and the rest of CFD group.

From Auckland and AUT and Aotearoa, The Land of the Long White Cloud, Graham Bidois for your leadership in the Certificate, and at AUT South Campus, for music spilling out of your office and your guns spilling our of your shirts, you are dearly missed; Jairo Gutierrez & Terry Brydon for welcoming me into AUT, always supporting me, directly and indirectly; Kerri Spooner for your fearless trail-blazing attitude; Jordan Alexander for your tireless enthusiasm for life and dedicated search to finding your tribe; Komala Sagadevan for your always kind words, especially those encouraging me to pursue the PhD, your soul has strength and courage; Alan Litchfield, Supervisor, collaborator, co-lecturer, LaTeX wizard, for your hours spent in meetings, you didn't flinch when it was suggested we develop blockchain courses, nor when I suggested blockchain scaling was a sticky wicket (I may still work on this yet). Its been enlightening watching you work, learning to be like a reed and bend with the wind, and watching you steer a research project with the slightest suggestion.

And to Harvey Theodore Coolidge for grinding for grinding's sake.

To Mom and Dad for paying my computer science tuition and grudgingly accepting when I dropped out, may this finally close that chapter; To Mom who didn't get to see this project to completion, know that I'll finally get that job; To Jennie, my favourite person, thanks for driving me to the airport while I packed in the parking lot and left home for an adventure that is still going today.

And lastly, to Donna, Lando, and Lincoln, thanks for a year of Sundays, we're just getting started.

Jeff
Te Atatu Peninsula, Auckland
September, 2023

Chapter 1

Introduction

> *This sounds like there's some kind of scalability trilemma at play. What is this trilemma and can we break through it?*
>
> Vitalik Buterin
> November 18, 2016

This Genesis Chapter presents the rationale of the study and the research objectives. It summarises the research questions, study significance, and main results, followed by the thesis structure.

1.1 Background & Rationale of the Study

The rise of computing in the second half of the twenty-first century brought with it a paradox of digital objects. How could you claim unique ownership and allocate value to something that could be copied quickly and efficiently, bit-by-bit, resulting in exact duplicates? In the realm of digital finance, the concept of digital abundance presents a unique challenge, particularly when applied to money. Unlike physical cash, which is inherently difficult to replicate due to its tangible nature and security features, electronic cash can easily be duplicated by adversaries. This ease of replication in the digital world poses a significant risk to the integrity of digital currencies. The advent of cryptography has enabled the creation of unique digital objects, an innovation with the potential to revolutionize the concept of currency in the digital age. Cryptography offers a way to secure information, making it possible to create digital assets that are unique and non-replicable. This characteristic is crucial for digital objects to function effectively as a proxy for currency.

However, even with the advances in cryptography, there remains a significant hurdle to overcome for the establishment of a truly decentralized digital currency: the double-spending problem. This problem arises from the question: How can we ensure that someone doesn't use the same digital dollar twice? In a decentralized system, without a central authority to verify transactions, it becomes challenging to prevent an individual from duplicating their digital currency and spending it multiple times. The double-spend problem was thought to be unsolvable in the realm of digital currency until a 2008 paper appears on the cryptography mailing list titled *Bitcoin: a peer-to-peer electronic cash system* (Nakamoto, 2008).[1]

Now known as a blockchain, the data structure that allows users to trust strangers with monetary transactions has experienced tremendous growth in the past decade. The technical details of a blockchain borrow many technologies from distributed computing and cryptography in the 80s and 90s such as cryptographic hashing, anti-spam email headers, linked time-stamping, and public key cryptography (Antonopoulos, 2017). The unique addition to prevent double-spending is the consensus algorithm that allows a linked-list representing the longest proof of computational work to represent the true state, or ledger.

In the years since 2008, distributed systems research has experienced a second-coming

1. `cryptography@metzdowd.com`; archived at `https://www.mail-archive.com/cryptography@metzdowd.com/msg09959.html`

and a new academic field of blockchain research has emerged. Many roadblocks appear as developers and researchers try to improve blockchains beyond the original Bitcoin specification. Today the Bitcoin blockchain is seen as slow, expensive, inefficient, archaic and clumsy; in essence atrophied (Croman et al., 2016). Nevertheless, the heart of the Bitcoin network continues to beat every ten minutes since inception, with no downtime, while securely allocating hundreds of billions of dollars.[2]

Bitcoin, which represents a decentralized network, and blockchain, the underlying data structure technology, constitute the primary element in the motivation cycle behind the present research, shown in Figure 1.1 and reviewed in Chapter 2.

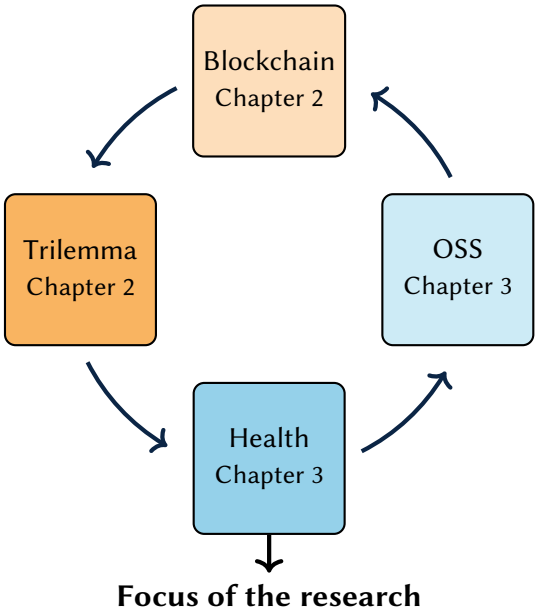


FIGURE 1.1: The motivation of the study comes from the study of blockchains which leads to the blockchain trilemma. These topics are reviewed in Chapter 2. This leads to the concept of software health in the open source domain, reviewed in Chapter 3. The motivation forms a loop as open source is a foundation principle of blockchain software.

As blockchain enters the lexicon[3] it becomes clear that many of the intentional design factors are also limitations to the growth of the network. The Bitcoin network is a method to transfer value between peers without intermediaries and this can handle at most around seven transactions per second (Xiao, Zhang, Lou, & Hou, 2020). If more transactions are required then people must queue for inclusion in the blockchain with preference given to those offering

2. The market capitalisation of Bitcoin as of writing in 2023 is around USD $500 billion.
3. In 2015 the Oxford English Dictionary added entries for blockchain and mining. This follows the addition in 2013 for Bitcoin.

the highest fees. In times of congestion, fees can spike and price people out of participating. Fees auctions are a prime factor in the scaling of networks. The peer-to-peer value transfer aforementioned relates to the property of the system being decentralised. Combining these two properties: scaling and decentralisation, with security, presents a trilemma.

The blockchain trilemma, originally proposed by Vitalik Buterin and widely acknowledged in the community (Buterin, 2016), serves as a pivotal element in the motivational framework depicted in Figure 1.1. This trilemma posits that a blockchain system can achieve only two out of three objectives: decentralisation, security, and scalability (Xiao et al., 2020). There are various proposals from both the blockchain and academic communities, aiming to mitigate this constraint. Often these suggestions reintroduce centralised elements inspired by distributed systems research. The lingering absence of a definitive solution to the blockchain trilemma could suggest several possibilities: either the issue is not critical enough to warrant research, the problem is intrinsically difficult to solve (indicating a stagnation in innovation), or there exists no solution at all. Given the plethora of emerging protocols, the issue appears to merit investigation. Whether a solution exists remains an open question, but resignation will not further the pursuit of an answer. This narrows down the viable interpretation to the middle option: the blockchain trilemma represents a challenging problem, or as framed in this study, progress in innovation has stalled.

In contemplating the role of innovation, the study arrives at the third motivational element within the research cycle in Figure 1.1. Questions emerge such as whether a state of health is requisite for a software project to innovate or succeed? Furthermore, the study probes whether the blockchain ecosystem is fundamentally unhealthy. These queries serve to guide the investigation towards its defined research objectives and questions. Although active research looks at the health of software teams, systems, and ecosystems, as detailed in Chapter 3, a conspicuous gap exists in the scholarly work regarding the health context, specifically within blockchain projects.[4]

There is one last motivation element to complete the cycle in Figure 1.1. Open source software (OSS) is a key scoping requirement in the study as it lays the foundation for distributed ledger technology to be born out of distributed systems. The open source nature of the blockchain community allows for easy migration and copying of projects that can be slightly altered and rebranded. Much of this in the spirit of looking for solutions and improvements to the trilemma.

4. In this work, the term *software health* refers to the health of an individual software project and is inclusive of software development within teams, activity and version control metrics, and resultant software artefacts. See Section 3.2.2.

1.2 Significance of the Study

The thesis focusses primarily on poorly understood concepts related to software health as applied to the fields of OSS and blockchain software. These projects are reliant on voluntary or freelance contributions where developers often migrate between projects based on individual motivations and incentives. This landscape renders a comprehensive understanding of software health even more crucial, to enable a holistic evaluation of projects that is not reliant on the contributions of individual developers.

Addressing a gap in academic discourse, the study provides a well-defined framework for assessing software health in the blockchain OSS context. Drawing upon empirical methods, it operationalises metrics capable of facilitating targeted resource allocation for project managers and meaningful engagement for contributors. As noted by Goggins, Lumbard, and Germonprez (2021), prior research has amassed indicators of OSS activity, but consensus on a coherent framework for health assessment remains elusive. Hence, the study serves a critical academic need by offering a systematic, reproducible methodology that advances the current literature.

Beyond academic contributions, the study's significance extends to its broad applicability for various stakeholders, including researchers, project managers, and governance bodies. By offering actionable metrics and insights into the unique ideological and incentive-based aspects of blockchain OSS, the study has the potential to inform both resource allocation and governance policies. This includes the standardisation of health metrics, which could be instrumental in guiding the allocation of community and industry grants.

1.3 Research Objectives

This study is driven by the complexities inherent in OSS, software health, and the scalability and consensus mechanisms of blockchain technology. Inspired by the proposition that advancements in blockchain performance are not solely rooted in technological evolution but are also closely linked to the health of the corresponding OSS project, this research provides a means to ascertain software health via the following objectives:

- To formulate a taxonomy that positions blockchain consensus within the broader scaling considerations posed by the blockchain trilemma.
- To clarify the notion of 'health' within blockchain-based OSS, laying the groundwork for empirical studies.
- To carry out an examination of diverse health indicators within OSS frameworks.

- To develop a theoretical framework capable of evaluating the health of blockchain projects within OSS.

These aims are refined into focused research questions, which are summarised next and investigated using methodologies detailed in Chapter 4.

1.4 Research Questions

Research questions serve as a guide for the study by helping inform the literature review, setting clear goals and objectives, and guiding methodology and methods. The following three research questions begin with inquiry into the blockchain trilemma (Figure 1.1), and move to investigating health in the context of blockchain OSS.

The first research question is motivated from the study of blockchains and the relationship of factors in the blockchain trilemma.

Research Question 1

What are the factors that influence blockchain consensus?

The second research question and its sub-questions spawn from the gap in the literature on software health.

Research Question 2

What is a definition of software health?

RQ2a

What metrics express open source software health factors?

RQ2b

What is the nature of the relationship between factors influencing software health?

The third research question ties the study of blockchain software together with software health.

Research Question 3

What is included in a comprehensive model of blockchain software health?

The research questions lead the investigation into the blockchain trilemma and software health to address the gap in the literature.

1.5 Contributions to Knowledge

This section summarises key artefacts of the thesis (illustrated in Table 11.1). A preliminary contribution pertains to the re-characterisation of the blockchain trilemma into a more focused dilemma. The dilemma navigates between consensus methods, intrinsically tied to decentralisation, and performance factors, which correlate with scaling capabilities. This is peer reviewed and published as *A Taxonomy of Blockchain Consensus Methods* (Nijsse & Litchfield, 2020) in Chapter 5. The taxonomy serves as part of the motivation (Figure 1.1) for examining the health of OSS in the blockchain domain.

Following this, the thesis introduces a definition of software health, drawing from cross-disciplinary literature in natural ecosystem health, business ecosystem health, and software ecosystem health (Chapter 3). Within this paradigm, the health of a software ecosystem is conceptualised as a composite of sustainability and robustness, additionally influenced by the factor of niche occupation. When considering individual software projects, health is further delineated into its constitutive components of sustainability and robustness. The sustainability dimension itself is operationalised through the constructs of general interest and developer engagement. This newly posited definition of health establishes the conceptual groundwork for the subsequent empirical studies and the formulation of the theoretical framework.

A theoretical framework for evaluating the health of OSS, specifically within the context of blockchain projects is presented in Chapter 9. This framework employs latent constructs—general interest, developer engagement, and software robustness—each operationalised via specific metrics. Interest is assessed through the observed variables of forks, stars, and mentions. Robustness is operationalised through the metrics: criticality, time since the last update, geographic distribution of contributors, and market capitalisation ranking. Engagement is captured through the number of authors, pull requests, commits, and comments.

This framework gains empirical substantiation through exploratory and confirmatory factor analyses. Specifically, the exploratory factor analysis for engagement (Chapter 6) is published in the proceedings of the Hawaii International Conference on System Sciences (Nijsse & Litchfield, 2023a). A confirmatory factor analysis (Chapters 7 and 8) focusing on the factors of engagement, interest, and robustness is currently under review (Nijsse & Litchfield, 2023b).

Thus, the framework provides methodologically rigorous and empirically grounded metrics that contribute to a comprehensive understanding of the health of OSS blockchain projects. The future research directions this highlights are in Section 11.4 beginning with validation outside the domain of blockchain software.

1.6 Thesis Structure

This thesis is organised beginning with the background literature that drives the motivation as shown in Figure 1.1 there are two literature review chapters corresponding to the two knowledge bases. Chapter 2 provides a review of blockchain technology, and, Chapter 3 examines literature related to the health of software projects, focusing on metrics and conceptual frameworks. In Chapter 4, the research methodology employed for data collection and analysis is detailed.

Results begin in Chapter 5, which presents a taxonomy of blockchain consensus methods. Chapter 6 and Chapter 7 present the exploratory factor analyses of software health, breaking down metrics related to engagement and other health dimensions. Building upon these findings, Chapter 8 introduces the structural equation model that models the relationship between the latent factors.

The theoretical contributions are next in Chapter 9, followed by Chapter 10 where the results are analysed in the context of both academic research and limitations. Finally, Chapter 11 summarises the key contributions of the thesis, discusses practical implications, and suggests directions for future research.

1.7 Conclusion

The background motivation is presented and shown in Figure 1.1 from which the research objectives are laid out. These objectives are refined in the research questions which are developed more thoroughly in the proceeding literature review chapters. The primary aim of this study is to improve the present state of the knowledge in software health by defining and modelling health within the scope of OSS blockchain projects. Next, Chapter 2 reviews blockchains as distributed ledgers beginning with distributed systems.

Chapter 2

Blockchain Literature Review

> *I've been working on a new electronic cash system that's fully peer-to-peer, with no trusted third party.*
>
> Satoshi Nakamoto
> November 1, 2008

Blockchains Are a Broad topic born out of distributed systems research, cryptography, game theory, and economics. The literature review encompasses these topics as they relate to blockchains, beginning with traditional consensus through to the decentralised version. The blockchain trilemma details the contrasting aspects of decentralisation, scalability, and security. Lastly, a research question related to blockchains is presented.

2.1 Distributed Computing

The rise of distributed computing in the 1970s and 1980s required techniques to allow for file access across many sites connected by a slow, unreliable network (Lindsay et al., 1979). This includes Ethernet, developed for Xerox's Alto computer for local networking in 1973 (Brock, 2023), to the AppleNet protocol developed for Apple's Lisa computer in 1983. AppleNet is later refined to AppleTalk for the Apple II which allows computers to be connected with a basic serial port adapter and requires no network configuration (Oppenheimer, 2004). Although user-friendly, if more than one person is attempting to write data to a network disk at the same time, problems arise as a write operation must be done sequentially and any reads that happen at the same time might not be aware of the state update.

This read-write example lays out the components of a distributed computing system: distinct nodes that can communicate with each other. The nodes must independently be able to achieve some computational goal and be connected; for example, between two or more workstations. This aligns with the definition in Gu, Wang, Hua, and Lau (2017, pp.3–4) that also states that the computational entities must be autonomous; they can obtain a result without human intervention. In the times before connected nodes communication would be manual: a technician could calculate a result at one terminal and deliver it to the scientist at another station.

The issues that can arise from a manual messenger have persisted in the digital realm. In *A Note on Distributed Computing*, Jim Waldo, a distributed systems researcher, identifies four main problems in distributed computing:

> The hard problems in distributed computing are not the problems of how to get things on and off the wire. The hard problems in distributed computing concern dealing with partial failure and the lack of a central resource manager. The hard problems in distributed computing concern ensuring adequate performance and dealing with problems of concurrency. The hard problems have to do with differences in memory access paradigms between local and distributed entities. (Waldo, Wyant, Wollrath, & Kendall, 1994)

Schroeder (1993) and Hadzilacos and Toueg (1993) say that partial failure and concurrency

are the defining problems of distributed computing. What these researchers are getting at is consensus is a hard problem in the nineties with issues of failure and concurrency being highlighted and continues to require academic research to advance the field and keep up with user demand.

2.1.1 Consensus

The goal of the distributed computing system is to do some calculation using information from distinct sources. For example, an aircraft can have three separate pitot-static pressure sensors to calculate airspeed (Federal Aviation Administration, 2012). The flight-critical data comes from sensors that are physically distributed in the aircraft which then need to be aggregated into a single reliable value for the pilot (Hammett, 2002). Some questions arise: What if a sensor goes offline? Or is acting erratically? Is the data needed in real time? Can a fault be detected? An algorithm that takes in data from multiple sources and comes to agreement is needed.

Consensus methods allow for a network with multiple nodes—banking ATMs, aircraft pressure sensors, YouTube servers—to communicate in a manner that maintains truth in the presence of faults (Attiya & Welch, 2004).

Traditional consensus is introduced here before transitioning to the decentralised paradigm which is the present-day case and applies to blockchains.

Replicated State Machines

When an update needs to be communicated to the system, such as a bank account balance after an ATM withdrawal, it is called a state update. The state represents the current picture of the entire system at the specified time, for example, the list of all customer accounts and balances. The goal of the state machine is to update truthfully such that a node cannot communicate incorrect data.

One strategy to handle updating the state involves replicating the database across multiple instances and communicating updates between them. The state of the system is a sequential list of commands that is replicated between all nodes in the network. Coming to consensus on the state means every node executes the linearly ordered log arriving at the same condition.

A consensus algorithm must maintain consistent copies of the state across all nodes and process updates proposed by any of those nodes. In database design, the transactions should adhere to ACID: atomic, consistent, isolated, and durable. An atomic transaction either happens or not. From the Greek for indivisible, even though a data transaction may involve many constituents, the system can only see a commit or abort. This guarantees a transaction does

not get halfway written (Anderson, 2020). It is consistent if all participant nodes commit the transaction and reflect the update. An inconsistent state would be where one node does not record the updated transaction. Isolated means that it can be parsed out of the set without affecting others. And durable is a one-way property; if a transaction needs to be reversed it must be via another transaction to undo the operation (Skeen, 1981).

CAP Theorem

Consistency, Availability, and Partition tolerance (CAP) is a best two-of-three scenario for database designers. Proposed by Eric Brewer (2000) and proved correct by Gilbert and Lynch (2002), database consistency means that multiple reads from different locations will yield the same value; an update must be available to all participants at the same time. Availability implies that an operational node will return a response and there will not be a period of inactivity. Lastly, partition tolerance means a response will still be received in the event of network communication failure. For example, the CAP theorem applied to MongoDB[1] results in sacrificing availability but the database maintains both consistency and partition tolerance.

The CAP theorem is notably different from the blockchain trilemma (Section 2.4) as it applies to crash-fault tolerant (not Byzantine) distributed systems, although logically both sacrifice one of the three properties.

Safety & Liveness

There are two properties that must hold for consensus: safety and liveness (Lamport, 1977). Safety means that two processors will agree on the same value, in addition to the value having been proposed by one of the processors. Agreement on that value is sometimes referred to as consistency, and a processor proposing a value as validity. Valid means the processor decided on a value and proposed it, although it does not necessarily need to be correct. Liveness is the property that any sent message is eventually delivered. A system that has stalled waiting for a message is effectively dead because it cannot make progress (Attiya & Welch, 2004). This can be thought of as "'something good' eventually happens during execution", where eventually is loosely defined as finite (Alpern & Schneider, 1985). Figure 2.1 shows the lexicon and its relations.

J. Gray and Lamport (2006) label the two safety properties as consistency and stability. Here, stability means that once a state (`commit` or `abort`) is reached, the node will remain in that state indefinitely. This is an extension of validity.

1. https://www.ibm.com/cloud/learn/cap-theorem

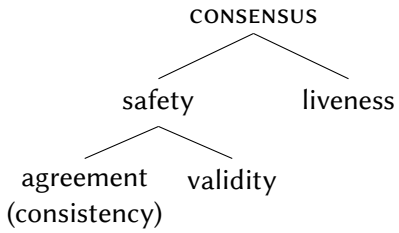


FIGURE 2.1: Consensus requires safety and liveness properties; further, safety is a combination of agreement and validity.

2.1.2 Crash Fault Tolerance – Honest Nodes

An important assumption in the description thus far is that all nodes are honest and trustworthy (the dishonest case, Byzantine, is in Section 2.1.3). Any messages that are delivered must be both appended to the log and committed. Should a node go offline or have some network interruption, then the remaining nodes will maintain the ledger until some point in the future when the faulty node recovers. This crash-fault tolerance is dominated by two- and three-phase commit protocols in the 80s & 90s and later by Paxos and its derivatives in the 2000s.

Two-Phase Commit

The Two-Phase Commit (2PC) protocol is developed independently by Lampson and Sturgis (1976) and J. N. Gray (1978). Inspection of the 2PC protocol can illuminate how independent processors can come to agreement in the presence of faults. The foundations lie in the *Generals Paradox* to which there is no solution.

Two generals are ready to march on an enemy and can only communicate via messenger. They both know that if they march together they will be victorious, however, if only one marches they will be defeated. Unfortunately the messengers can sometimes become lost, thus creating unsure lines of communication. The problem is to create an algorithm to guarantee the generals will act in unison (J. N. Gray, 1978).

In this scenario there is no fixed length communication strategy. Assuming the last delivered message gets lost, then one general is always unsure of the action. General A sends "attack at dawn" and if this message gets lost, then General B waits. If General B receives "attack at dawn" they must agree and send back "okay", only to have this message get lost leaving General A unsure if the message was received. So lets send a third message from A back to B: "confirmed". If this third message is lost, then its back to the first case. Continuing in this manner there is no algorithm to overcome this paradox.

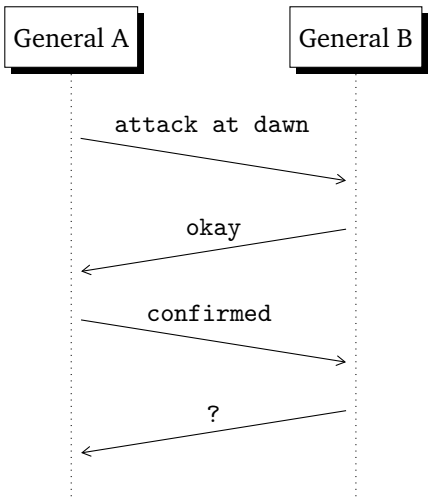


FIGURE 2.2: The Generals Paradox. If the last message gets lost the generals will never have assurance their actions are coordinated.

Given that the generals are nodes in a distributed system, 2PC is a method that replicates the state assuming reliable communication between participant nodes and a coordinator node.

In the first phase, called a PREPARE phase, a coordinator will send the transaction to the participating nodes and await their response. Each node can either agree with the transaction and `commit`, or reject the transaction and `abort`. The node responds with `yes/abort` to the coordinator. Figure 2.3 diagrams the process with messages `A` & `B` representing the PREPARE phase. The second phase, COMMIT, (messages `C` & `D`) involves another round-trip. After the coordinator tallies the responses they send the result to the nodes. If a single `abort` is received the coordinator will abort the process. Each node then aborts and waits for the next round. Lastly, if all nodes send back a `yes` the coordinator will let everyone know they can commit, and the node acknowledges an update to their state.

In the event of a site-failure when a participant is in a prepared-to-commit state (between messages `B` and `C`) this node is now blocked from proceeding as it must wait for confirmation to `commit` or to `abort`. At this point it can no longer continue processing updates, possibly halting the use of resources at the node's site as well. For this reason 2PC is called a blocking protocol. Once the coordinator restarts it will consult with the log and pick up where it left off. Should a participant node crash before writing to their log the coordinator will abort under timeout conditions. Should a node crash after logging, upon restart they can read the log and assess based on the coordinators decision.

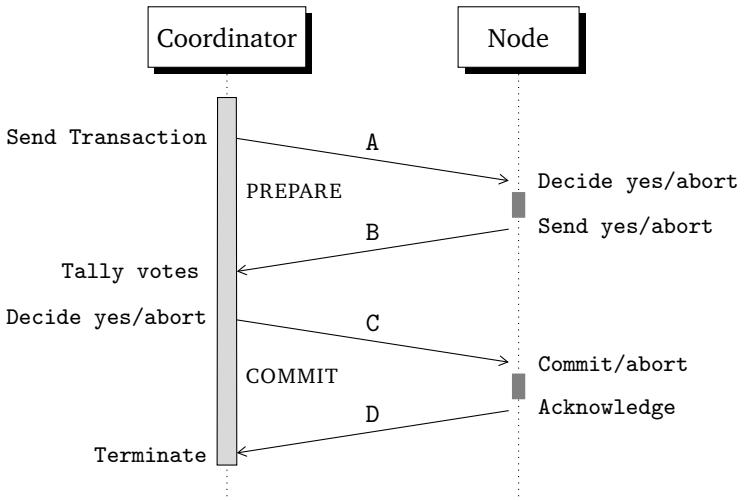


FIGURE 2.3: Two-Phase commit protocol. Messages `A` & `B` are the PREPARE phase, and messages `C` & `D` are the COMMIT phase.

Three-Phase Commit

The blocking behaviour described above motivated the move to non-blocking atomic commit protocols; Three-Phase Commit (3PC) is published by Skeen (1981) and involves an additional round-trip of messaging. Figure 2.4 shows how to prevent this blocking behaviour by adding a PRE-COMMIT, or buffer phase before the COMMIT phase.

The extra buffer phase separates the `commit` and `abort` states. In 2PC, given a failure after the PREPARE phase, it is unknown if the transaction is to be committed or aborted because both state transitions are reachable. In 3PC if a node is in the PRE-COMMITTED phase it can assume that the transaction will `commit` because the abort instruction will not be received until the final phase. In this manner when the coordinator fails the remaining nodes can collectively decide to `commit` if any one of them are in a PRE-COMMIT state and `abort` otherwise. When a node (not the coordinator) times out, they must communicate with others to determine the latest state as they cannot assume their own pre-committed state is accepted as the canonical transaction.

3PC has not been implemented in any industrial scale database due to the message complexity; the cost to wait for every node to communicate three times is too inefficient (Al-Houmaily & Samaras, 2016). Additionally J. Gray and Lamport (2006) "know of no [3PC] that provides a complete algorithm proven to satisfy a clearly stated correctness condition."

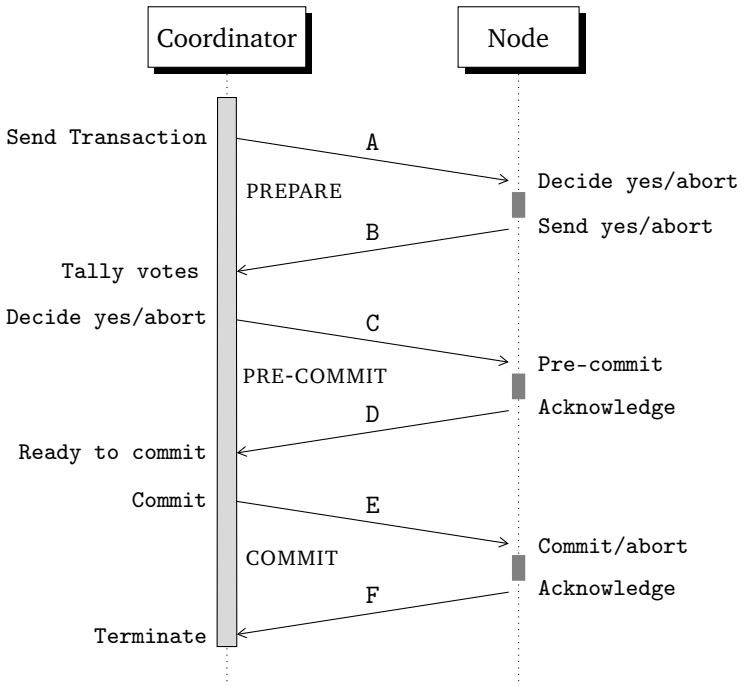


FIGURE 2.4: Three-Phase commit protocol. Messages C & D are the PRE-COMMIT phase, which allow for nodes to record the `pre-commit` in their logs.

Paxos

Paxos[2] is an evolutionary variant of 2PC of particular note and is mentioned before transitioning to the decentralised regime involving malicious actors. The algorithm achieves atomic commit using multiple coordinators and can continue to make progress as long as a majority of coordinators are responsive (Lamport, 1998). Paxos and its derivatives such as multi-Paxos (Renesse & Altinbuken, 2015) and fast-Paxos (Lamport, 2006) can all survive f faults given $2f + 1$ nodes.

Google's *BigTable* (Chang et al., 2006), and Amazon's *DynamoDB* (Decandia et al., 2007) are both developed based on Paxos and run at least five geographically distributed replicas. ZooKeeper (The Apache Software Foundation, 2019) is another Paxos-based system; maintained by the Apache Software Foundation and found in many popular services by companies such as Meta (Facebook), Yahoo!, and X (formerly Twitter) (Apache ZooKeeper, 2019).

2. *The Part-Time Parliament* by Lamport (1998) is the name of the paper inspired by the fictional parliamentary system on the Aegean island of Paxos where parliament would remain in session and the politicians would come and go, analogous to nodes arbitrarily going offline.

2.1.3 Byzantine Fault Tolerance

If a node behaves in an arbitrary manner it is unreliable and can be assumed in the worst case to be acting as a malicious adversary. This is known as Byzantine behaviour named after the Byzantine Generals Problem[3] formalised in the seminal paper by Lamport, Shostak, and Pease (1982) and inspired by NASA-funded work on fault-tolerant aircraft control systems (Wensley et al., 1978).

An extension of the Generals Paradox (Section 2.1.2), now several generals of the Byzantine army are camped near an enemy position. Each general commands their own battalion and can communicate via messenger. The generals must come to a decision to either attack or retreat while considering that there could be traitors among them. The generals must have an algorithm such that all loyal generals reach the same decision and any traitor(s) decisions can not change the outcome.

Reaching Agreement in the Presence of Faults (Pease, Shostak, & Lamport, 1980) shows for the general case that given m faulty processors that always lie, the number of honest processors must be: $n > 3m + 1$. This means that up to one-third of the processors can deliver arbitrary information with a lower bound of four in the system.

Early work to overcome Byzantine fault tolerance (BFT) is either too inefficient or could not allow for messages to be delayed for arbitrary lengths of time. The landmark result by Fischer, Lynch, and Paterson (1985) concludes that agreement among processors in which at least one is faulty is impossible without some form of timing assumptions such as a message timeout. This is because it is impossible to know if a message is delayed indefinitely (it may never arrive). The difference between a slow node and a crashed node is difficult to detect.

Practical Byzantine Fault Tolerance

In 1999 Castro and Liskov publish a milestone paper called *Practical Byzantine Fault Tolerance* (PBFT) that guarantees safety and liveness for up to f Byzantine nodes of $3f + 1$ replicas. This is in an asynchronous environment[4] meaning that there is no known bound on when messages are delivered. Importantly, Castro and Liskov's PBFT is practical and there are many implementations in diverse areas: fault-tolerant distributed storage, certificate authorities, secure multi-party computation, the Hyperledger blockchain project, and reputedly in the Linux-based systems SpaceX uses to dock the Dragon capsule with the International Space

3. Lamport originally stole the name from the Chinese Generals problem (also called the General's Paradox, Section 2.1.2) and wanted to name it The Albanian Generals Problem, but settled on the fallen empire to avoid offending anyone. See `https://lamport.azurewebsites.net/pubs/pubs.html#byz`.
4. Technically PBFT has weak-synchrony which is why it is shown in Table 5.3 as partially synchronous (Cachin, 2010). (Also referred to as eventual synchrony.)

Station (Edge, 2013).

The methods referred to above fall into the category of classical consensus for a distributed system and can be demarcated in time up until 2009. In the intervening years a number of initiatives arise to apply distributed systems knowledge to one specific problem: digital cash.

2.2 Digital Cash

Digital cash is, at its heart, a distributed systems problem. There are many attempts, summarised here, to decentralise fiat banking, create unique digital objects, and build a secure peer-to-peer (p2p) value transfer system.

DigiCash Called untraceable electronic cash, DigiCash is the decade-long project of cryptographer David Chaum, known as the father of digital cash. His scheme uses blind signatures (Chaum, 1983) which mean you cannot reuse a digital coin without revealing your identity and being subjected to blacklisting. DigiCash is not decentralised and requires an issuing authority to create notes and maintain transaction activity (Chaum, Fiat, & Naor, 1988). Active in Europe for a short period of time, DigiCash does not gain enough adoption to survive.

BMoney (WeiDai, 1998) & Bitgold (Szabo, 2008) Two independently proposed essays opining that a proof-of-work system can be used to create the money in the first place then use it to authorise transactions. However, within these proposals it is unclear how to resolve Sybil attacks or disputes that may arise.

PayWord and MicroMint Two simple micropayment schemes designed to handle small transactions on the internet that require a significant gain in computational efficiency to be practical. Both schemes are centralised and do not evolve into a product (Rivest & Shamir, 1996). Micro-transaction capability remains a leading use case for blockchain technology.

Hashcash Created by Back (2002)[5], Hashcash uses the idea that a hash function also represents random outputs. So a user can attempt multiple inputs until finding a hash that meets a requirement. This then represents a token in a value transfer system. Back has stated that Bitcoin is an extension of the Hashcash system, however, Hashcash is not a fully functional value transfer system and could not prevent double-spending.

5. The software is released in 1997, the technical draft is published in 2002.

Compact E-Cash This is a cryptographic scheme for off-line anonymous e-cash with extensions to provide traceable coins without a trusted third party. Once a user attempts to double-spend, all of their previous transactions become visible, thus de-anonymising their transactions (Camenisch, Hohenberger, & Lysyanskaya, 2005).

These precursors all contributed to distributed computing and cryptography research leading to the development of Bitcoin as a viable digital implementation of cash.

2.3 Blockchains

Applying consensus in the permissionless setting is what sets blockchains apart from databases. Removing the centralised server from distributed computing presents a different picture. In a peer-to-peer network every participant is at the same level of the hierarchy. A permissionless network allows anyone to join, participate, and leave at any time. Figure 2.5 shows a state diagram describing the iterative process of updating the state that is managed by the consensus method.

2.3.1 Decentralised Consensus

A traditional consensus method (centralised) and a distributed consensus method (decentralised) have an important distinction. The question: Who gets to propose updates? becomes critical because the permissionless nodes do not have trusted identities. Blockchain consensus is not possible until Nakamoto (2008) introduces the concept of proof-of-work (PoW) mining, where the process of proposing updates is accomplished by searching for cryptographic hashes. The winner of this race condition can append the next block to the chain and is rewarded for doing so. In the short term, this winner has maintained consensus. Compared to the centralised consensus of Paxos and PBFT, a known leader proposes updates which others may vote on. These votes constitute the equivalent resource to PoW clock-cycles.

In addition, these decentralised algorithms that answer: who proposes? and how-to-update? must be fault-tolerant like centralised systems, and resilient to Byzantine behaviour. Many alternate methods arrive on the scene in the past decade and a taxonomy is presented in Chapter 5.

Thus, a blockchain implements a distributed and democratic consensus algorithm that, in its first draft, is used for a value-transfer ledger.

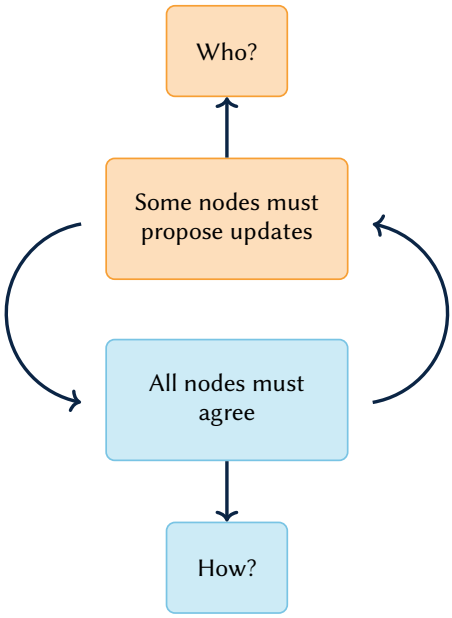


FIGURE 2.5: A state diagram for a distributed blockchain. The process of generating new blocks and updating the chain is a continuous loop. A consensus algorithm answers the questions of: How? and Who?

2.3.2 Bitcoin

Announced on a cryptography mailing list[6] in late 2008, the Bitcoin whitepaper (Nakamoto, 2008) is a brief description of how peer-to-peer electronic cash can work. Shortly after, the author, Satoshi Nakamoto, mined the first block, thus instantiating the Bitcoin blockchain on January 3rd, 2009. According to the design, the features of the Bitcoin protocol include: immutability, append-only behaviour, transparency, non-repudiation, and censorship resistance. Additional features that a p2p cash system could lend to its users include: a fixed and auditable money supply, divisibility well beyond one-hundredth of a unit (cents), decentralised governance, and a global reach.

In addition to the digital cash proposals described in Section 2.2, Bitcoin draws on many previous technologies. The data structure now known as the blockchain uses both linked time-stamping and Merkle trees which rely on hash functions. Linked time stamping is explored in a series of papers in the nineties by Haber and Stornetta (1991) as a method of verifying when electronic documents are published. When an author creates a document the hash of the document is signed with a timestamp and includes a previously published signed document.

6. `cryptography@metzdowd.com`; archived at `https://www.mail-archive.com/cryptography@metzdowd.com/msg09959.html`

The next one that is created is signed along with a hash of the present one creating a chain.

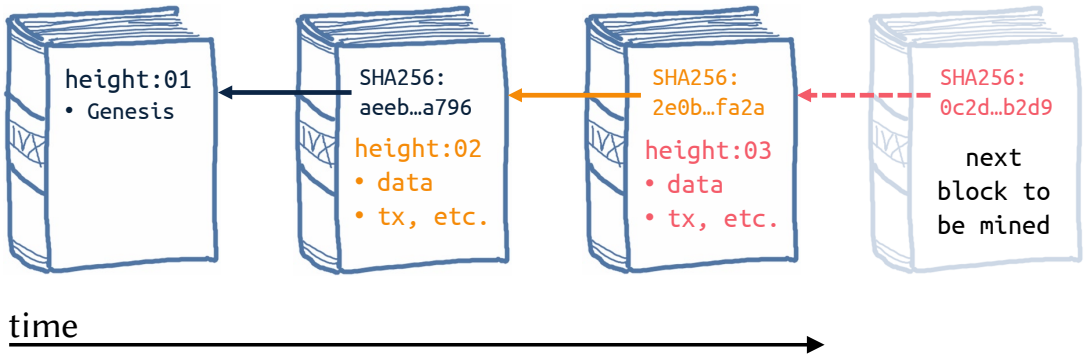


FIGURE 2.6: Data structure for a blockchain. New blocks are added containing a cryptographic hash of the previous block. Each block is analogous to a ledger recording transaction activity.

The use cases of cryptographic hash functions are extensive. A hash function takes a variable input such as a financial transaction or a electronic document and produces a fixed-length output such that given the output, no input can be deduced in a reasonable amount of time, and given multiple inputs, there does not exist a singular output (Stallings, 2017). Bitcoin uses the `SHA256` hash function (National Institute of Standards and Technology, 2012) both as a PoW and to improve efficiency in the data structure. Using hashing in this manner derived from anti-spam email header PoW, originally conceived by Dwork and Naor (1992) and independently by Back (2002), these hash challenges are designed to limit email spam by having the user expend some computational work in finding a partial collision of a hash function. The computation expended by searching for hashes is non-reversible and represents a scarce resource in the form of clock-cycles. This computation is feasible for regular users but becomes expensive for spammers.

A replicated data structure that stores abundant information such as financial transaction data will quickly become bloated without an efficient use of space. A Merkle tree, named after cryptographer Ralph Merkle, contains the data in the leaves with any internal nodes being hashes of the children (Merkle, 1980). This allows for efficient retrieval and immutability; if a child node changes, then all the parent nodes back to the root will also change. In cryptography this is called a cryptographic commitment. Any unauthorised changes can quickly be identified and rejected.

To this point not much is said about the identities of participants in the network. Asymmetric public key cryptography allows for public keys to be proxies for participants. Bitcoin identities are public/private key pairs whereby a user must sign transactions with their pri-

vate key. Digital signatures allow a user to sign a document with their private key such that anyone with the public key can verify the signature (Rivest, Shamir, & Adleman, 1978; Merkle, 1980). Bitcoin uses an elliptic curve cryptosystem for address and key generation (D. Brown, 2010) which has a slight benefit of smaller key sizes when compared to the well known RSA scheme (Rivest et al., 1978). Importantly this does not remove the human from the transaction; the use of public key cryptography only adds a single layer of obfuscation. Most cryptocurrencies are considered pseudonymous because chain analysis can be used to track users (Narayanan, Bonneau, Felten, Miller, & Goldfeder, 2016).

The revolutionary contribution of Nakamoto is to use PoW to secure the ledger against double-spending attacks. This defined a new method of distributed consensus, now known as emergent consensus, or eponymously as Nakamoto consensus. Participants are insured against their coins being spent twice by the emergent property of the longest chain. If there is only ever a single chain, it is the longest and represents the most proof-of-work; the canonical state of the blockchain. As more blocks are added to the chain there is an exponential decrease in the probability of a block being rejected by other nodes. This is the method that secures the blockchain against malicious nodes attempting to double spend. Nakamoto consensus resolves forks in the chain by allowing both branches to remain active (alive) until there is a clear longest chain. At this point the shorter branch is orphaned and any outstanding transactions need to be reprocessed and may be superseded by others paying higher fees. This is the general process by which all proof-of-work-style blockchains maintain consensus, known as the longest-chain fork-choice rule. Further description and analysis of blockchain consensus is the topic of Chapter 5.

Proof-of-Work

There is an important distinction where Bitcoin varies from other methods of reaching consensus. There are incentives for nodes to act honestly that are built into the protocol. The first is called the coinbase transaction and awards freshly minted bitcoins to whoever added the block to the chain. This is how new bitcoins come into circulation. The second is from transaction fees. By listening to the network, validating transactions, and including them in a block, whoever is operating the node can choose to include transactions that offer an extra fee. Because Bitcoin itself is designed to be digital money, this makes perfect sense and is why cryptocurrency is the original use-case for a blockchain.

Consensus rules maintain the longest chain and reward participants which also acts as a Sybil resistance mechanism. In Figure 2.7 miners compete to win a hashing competition and propose updates to guide consensus. A miner is a network participant that contributes their computing power in a demonstrable way. A fair way to allocate the incentives would

be by some resource that can not be gamed or monopolised. One such way is by computing power as proposed by Dwork and Naor (1992) in relation to email spam, and refined by Back (2002) for digital currency. Bitcoin miners participate by using their hardware to validate transactions and suggest new blocks. For this effort they receive rewards in proportion of their contribution to the network as a whole. Bitcoin uses a `SHA256` hashing algorithm as the hash puzzle that miners have to find a solution for to be able to publish a block.[7]

```
Block = Hash(nonce||previousHash||data||data||...||data) < target
```

Hash function output has a random distribution and so to find a block, your hash must be below a certain target level. The target level is the hash as a hexadecimal number, the order of magnitude can be seen by the number of leading zero bits, for example:

```
0000000000000000000117c467ab5336077cb04f7f70ea6ebcd68e0b3ef6cf909
```

is the successful hash of block `529283`. The only way to find a hash with a smaller value than the target is to change a nonce value and re-hash the bundle of transactions over and over. When a target is hit, the block is broadcast to the network as a proof of computational work done in winning the hash competition.

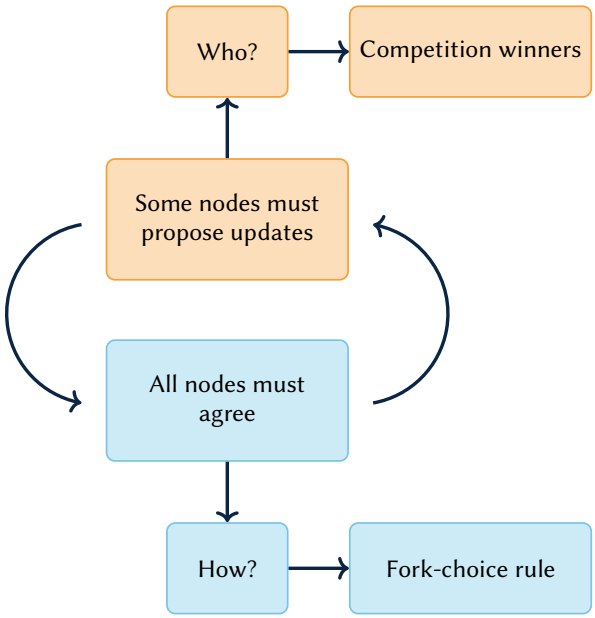


FIGURE 2.7: A state diagram for a distributed blockchain showing the hashing competition for the process of generating new blocks and updating the chain.

7. Hash *Puzzles* are a misconception in Bitcoin. There is no pre-defined solution or optimisable method as a puzzle would suggest, it is simply a game of chance.

The bits here should be randomly distributed, like a lottery, to prevent gaming the system and earning more rewards than your proportion.[8] Computing cycles in the Bitcoin network are called hashpower in reference to `SHA256`. As more miners come online, the total hashpower increases leading to greater overall probability of successfully hashing a value below the target. To keep the temporal distribution of blocks even, this target difficulty automatically adjusts according to the protocol every 2016 blocks, or approximately two weeks.

The difficulty adjustment aims to keep the time between blocks (successful hashes) at around ten minutes. In the early days Satoshi and a few others could use their PC processors to find blocks every ten minutes. The hashpower has steadily increased and so has the difficulty target to keep the block time constant. Finding a hash of a block that is below the target size is a discrete event; it is either below or it is not. As with a lottery, it is only a matter of time before a hash is found, and the previous hash is independent of the current attempt. Statistically, this is a Poisson distribution.

A brilliant idea that Satoshi incorporates into the Bitcoin protocol is that the number of newly minted bitcoins decreases over time. Every 210,000 blocks (≈ four years) there is a halving event and the next block found is only allowed to pay out half the bitcoins. This defines a finite money supply. If you run the clock forward, assuming a new block is added every ten minutes, there are no new bitcoins minted after the year 2140. This is the main incentive for miners to win a block. According to the present protocol, there will never be more than 21 million bitcoins which makes it a deflationary currency.[9]

Bitcoin's Limitations

The limited capabilities are quickly recognised when Bitcoin is put into practice and exposed to the scrutiny of the open source community. Technically it was unproven if the emergent (Nakamoto) consensus algorithm could work; there was no formal verification in the academic community. A milestone result is the J. Garay, Kiayias, and Leonardos (2015) paper showing that the Bitcoin consensus protocol holds up given a synchronous network assumption (Pass, Seeman, & Shelat, 2017; J. Garay et al., 2015; J. A. Garay, Kiayias, & Leonardos, 2019). As a new regime of consensus protocols, few researchers are familiar with proofing methods to verify a consensus algorithm, and many still exist without a formal verification.

Users of Bitcoin have their criticisms of the functionality as well. Its slow compared to

8. As the Bitcoin network has matured, dedicated hardware—Application Specific Integrated Circuits (ASICs)—to solve the `SHA256` algorithm have dominated. It is no longer feasible for a single participant to mine bitcoin without dedicated hardware.

9. Technically, slightly less. Due to the issuance schedule being a decreasing geometric series and the smallest unit, a satoshi, being 1/100 million, there will only be 20,999,999.97690000 million bitcoin mined (Antonopoulos, 2017).

ordinary payment (and distributed) networks. It could take up to ten minutes for a transaction to be confirmed and if the network is busy, or the difficulty has adjusted upwards, this could be much longer. Others have tweaked the algorithm in the form of software forks to create, for example, Litecoin[10] and Dogecoin[11] among many others. Additionally, Bitcoin does not have added privacy features allowing only pseudo-anonymity (Narayanan et al., 2016), its stack-based scripting language is limited, there is restricted ability for multiple parties to author transactions (Antonopoulos, 2017), has no natively optimised storage, and is still living in the shadow of the Silk Road (Christin, 2013), Mt. Gox, or more broad industry events like the Terra-LUNA and FTX collapses.

Most of these limitations could be put under a wish list of features that Bitcoin does not offer and that show up as benefits in other projects. The most impactful advance to date is that of programmability which is a core tenet of Ethereum.

2.3.3 Ethereum

Ethereum is the first major alternative blockchain implementation to gain traction that is not a fork of Bitcoin. The Ethereum co-founder is working on Bitcoin in 2014 and realises the limitations in the scripting language so decides to code up a new blockchain (Buterin, 2013). Shortly after, a formal specification for the Turing-complete language and virtual machine is released (Wood, 2014). In addition to being able to execute computation, Ethereum uses a Proof-of-Stake (PoS) consensus method (Buterin & Griffith, 2017). PoS is where users lock up a portion of their tokens to secure the network earning rewards proportional to their stake. In 2022, Ethereum hot-swapped the main chain to PoS Beacon Chain without missing a single block.[12]

Turing complete computation is accomplished via smart contracts that are executed in a virtual machine, called the Ethereum Virtual Machine (EVM). The EVM operates independently of the underlying hardware, ensuring deterministic computation that yields the same result across all network nodes. Each full node runs a copy of the EVM to verify transactions and smart contract executions, playing a crucial role in the decentralisation and security of the Ethereum network.

The incentive model of Ethereum is unique by having multiple layers: gas to be charged by the network for computation and ether to be rewarded to the miners for validating blocks, running code, and maintaining the chain. Finance applications are the most prominent on

10. Technically Litecoin is a fork of Fairbrix which is a fork of Tenebrix which changed Bitcoin's PoW algorithm from SHA256 to Scrypt (Song, 2019).
11. Dogecoin is a fork of Luckycoin, which itself was a fork of Litecoin.
12. Ethereum is bootstrapped as a PoW protocol using a different algorithm than Bitcoin–called *ethash* which is more memory-hard than SHA256.

the chain, for example fundraising by initial coin offerings (ICOs) (Poon & Buterin, 2017), and decentralised exchanges, and decentralised marketplaces.

Ethereum's Limitations

Ethereum has its own limitations distinct from Bitcoin, and in the years since the network's release there are a number of competitors attempting to improve upon it, known as alternative layer 1s. Storage, bootstrapping, network congestion, and privacy are all well known areas for improvement.

The Ethereum network can handle approximately 15 transactions per second which has been tested numerous times. Network congestion first became a vibrant issue when a game, *Crypto Kitties*, where users trade and breed unique digital avatar cats, drives gas prices up. At busy times it is expensive to use Ethereum, where the fees-based model prices out many average users. Ethereum gas prices are an auction on block space, and so when the network is popular there is a spike in fees that can exclude others, human and non-human.

The complexity allowed by storing and executing solidity code in the global state presents a large attack surface for security holes to emerge. The DAO hack and the various other smart contract failures are examples of such vulnerabilities (Casino, Dasaklis, & Patsakis, 2019; Atzei, Bartoletti, & Cimoli, 2017). Events like this have in turn sparked debate about open source software governance which is linked to how decentralised the network can be (Section 2.4.1).

2.4 The Blockchain Trilemma

The research motivation loop in Figure 1.1 leads from the study of blockchains to the blockchain trilemma (Buterin, 2016). A triad of security, scaling, and decentralisation, the trilemma is a well known trade-off when designing a decentralised distributed system. In Figure 2.8 the trilemma is such that a choice must be made by the developer of which property to sacrifice.

Two parts may be well designed, but the third will suffer. Xiao et al. (2020) says "Depending on the application scenario, a desired protocol needs to strike a balance between three metrics: decentralisation, security, and scalability." Although this fits with the general narrative in the community it simplifies many details such as how are these metrics assessed? and most importantly, can a balance be struck with security?

2.4.1 Decentralisation

Communications networks can take the forms shown in Figure 2.9 where the dots represent nodes and the lines are connections between the nodes. A centralised network has a single

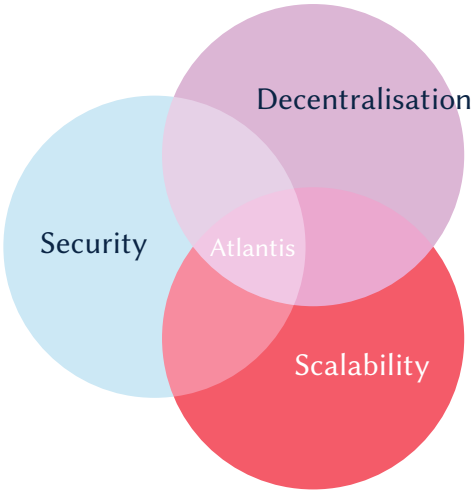


FIGURE 2.8: The blockchain trilemma: picking any two of three ideal properties compromises the third.

point of command and control, but also is vulnerable to failure; should the home node go down, the network is paralysed. For redundancy nodes can be added (often replicated) at additional sites. Removing reliance on any single node or cluster of nodes results in a mesh network. In practice, most networks are a combination of the two, "Such a network is sometimes called a 'decentralised' network, because complete reliance upon a single point is not always required" (Baran, 1964).

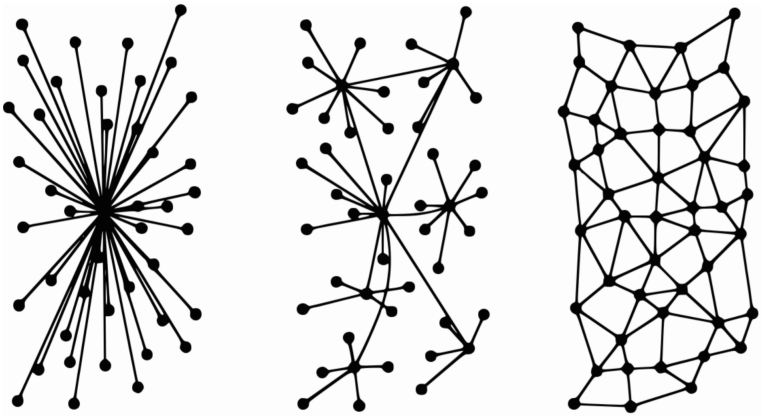


FIGURE 2.9: A centralised, or star, network [left]; decentralised [centre]; and distributed, or mesh network [right] (Baran, 1964).

The above description is for an electrical communication network which can equally be applied to an optical (internet) or electromagnetic (mobile) network. A decentralised blockchain

has a subtle distinction here because the blockchain operates on top of this communication infrastructure.

Decentralisation of a blockchain refers both to the use of a mixed communication network and the absence of a centralised authority. Large scale OSS projects exist in a continuum between centralised top-down control at one end and decentralised user-guided at the other. Centralised software projects are the standard, for example the Ripple Company[13] has a hierarchical structure and makes decisions in a top-down corporate style. Additionally clients must have permission to join and participate in the Ripple network. These are standard properties of enterprise blockchains.

Decentralised software projects are more difficult to pin down and define. This is because in the beginning the project is started by a small group or single person and as it grows the future direction is handled by the community as a whole. Achieving consensus in a social setting can be contentious and involve lengthy debate and many false starts (Bier, 2021). Bitcoin is started by a single person (presumably) and presently has hundreds of developers contributing to the codebase (which defines the governance) and numerous software implementations. A key feature of a decentralised network is that participants are free to join and leave; anyone (human or autonomous) can use Bitcoin at any time. This includes editing the software. Proposing, writing, and publishing a new feature to the code must be agreed upon by the community. Even innocuous things like commenting on the code can take months to have committed.

Decentralisation Link to Consensus

In the architecture of a blockchain network, the role of consensus mechanisms is pivotal for ensuring decentralisation. Decentralisation manifests on a continuum, where its degree is intrinsically linked to the requirements of the consensus algorithm employed. Centralised networks mainly require crash fault tolerance as outlined in Section 2.1.2, with the assumption that the network is devoid of malicious actors. On the contrary, highly decentralised networks necessitate consensus algorithms that can reach agreement even in the presence of Byzantine faults. These algorithms are all BFT variants, see Section 2.1.3. This need for BFT in highly decentralised networks is corroborated by cluster analysis, mapping the terms Byzantine and consensus closely together, as visualised in Figure 2.10.

A textual analysis is completed on the set of survey papers used to create a taxonomy of consensus methods (Chapter 5) using *NVivo* (2018). Of the top thirty terms of minimum word length, four are clustered combining closely related meanings and synonyms. Terms such as

13. Ripple operates an enterprise blockchain payments network which is distinct from from the Ripple cryptocurrency $XRP.

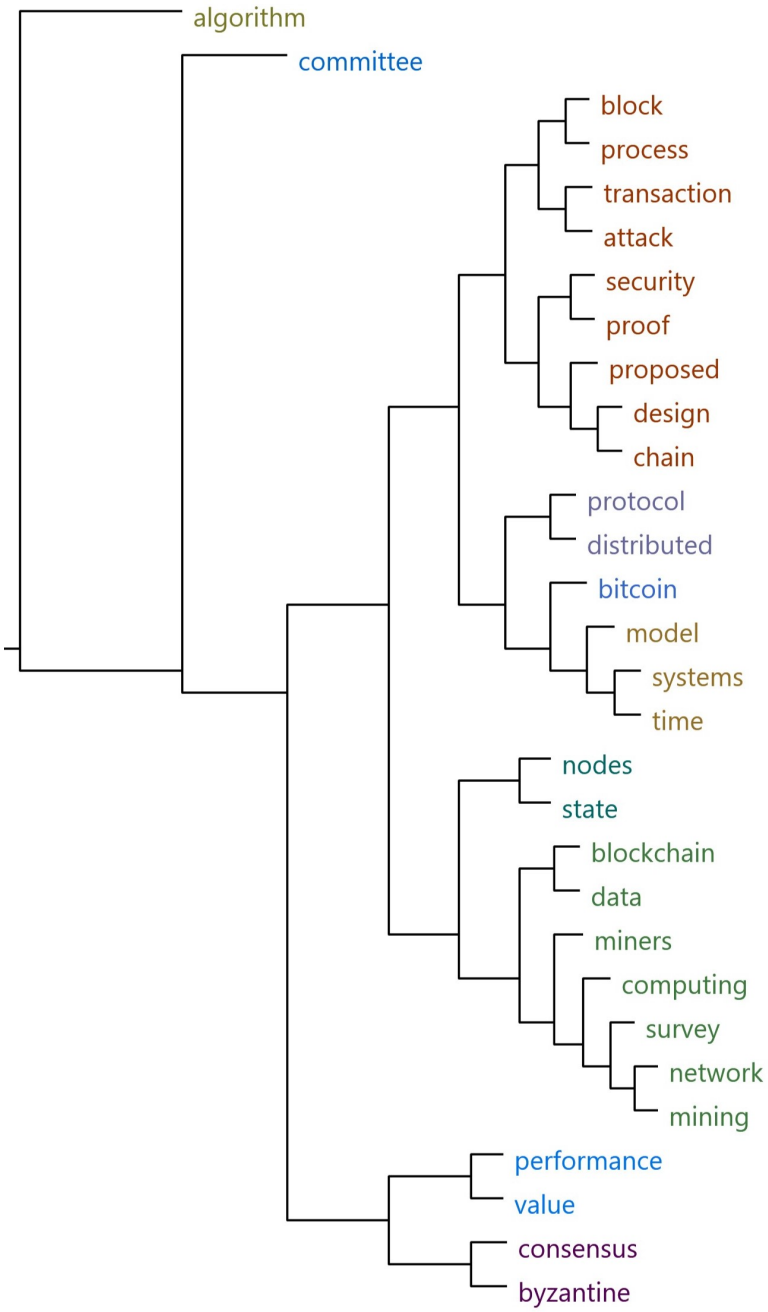


FIGURE 2.10: Cluster analysis of blockchain consensus surveys. *Performance* and *consensus* are at the same level in the tree and isolated from the other terms. The top thirty terms are identified using *NVivo Plus 12*.

number and synonyms such as *amounts, list, total* are excluded as they are more linguistic and not relevant to blockchain design. At the same level the terms *consensus* and *Byzantine* are in close proximity. These two binary branches are distant from the rest of the terms indicating they are not influenced by the other high frequency words. Figure 2.10 shows the complete tree. Note that *algorithm* is at the root of the tree because it does not consistently get mentioned in conjunction with other terms. *Performance* and *value* are coupled together and also colour-coded together. The grouping of consensus and performance at the same relative level indicates that within the research literature these are linked. Accordingly, the consensus algorithm is a major contributing factor to performance and scalability of the protocol; both factors are limiting blockchain development and adoption (Bano, Sonnino, et al., 2017).

Therefore, it can be concluded that the degree of decentralisation from the blockchain trilemma in a blockchain network is intricately associated with the type of consensus algorithm that is engineered to sustain state coherency among potentially adversarial nodes. Additionally the ability to scale a blockchain is linked to the performance.

2.4.2 Scalability

Scalability is not one property of a system, but a term that relates several quantitative metrics to each other (Croman et al., 2016). Fully decentralised applications are good at horizontal scaling by increasing the number of nodes in the network up to many thousands or tens-of-thousands because anyone has permission to run the client (Vukolić, 2016). In a decentralised system vertical scaling is managed individually on a per-node basis, where as long as the node meets minimum requirements it can run the client. If the requirements are too high or expensive, decentralisation will be compromised. These network scaling characteristics are not to be confused with network activity where issues such as throughput and latency are seen to be a bottleneck to scalability (Bano, Al-Bassam, & Danezis, 2017).

For a blockchain to be useful, information must be exchanged between participants (autonomous or otherwise), often called a transaction. Two design decisions fundamentally limit the number of transactions the network can process, dictating a trade-off and limiting the capacity as measured in transactions per second (TPS). For example, in Bitcoin the block-time and the block-size are hard-coded to ten minutes and four million weight units[14] and a Segwit[15] transaction size is 565 weight units so the upper limit is $\frac{4000000}{565} = 7079$ Segwit

14. This is approximately 4 million bytes (4MB); see `https://en.bitcoin.it/wiki/Weight_units`.
15. Segwit is segregated witness which is a more efficient transaction type introduced in 2017.

transactions per block. The theoretical block time is 600 seconds[16] and so

$$\text{throughput} = \frac{7079}{600} = 11.8 \text{ TPS}.$$

In practice there are different types of transactions that have more advanced structure and take up more weight so the throughput observed is closer to 5.79 TPS in the latter half of 2023.

In a geographically distributed network there must be time for a block to flood the network, allowing for consensus, before the subsequent block is found. Compare with Ethereum that has a much shorter blocktime of 12 seconds, yet can only handle around 15 TPS on the base layer, and network fees increase during busy times.

Scalability of transactions is the big problem in terms of using blockchains in the future; the global economy requires hundreds of thousands of TPS. Considering a few use-cases:

- Money in the traditional sense as a medium of exchange (not store of value) the VISA network averages over 1,700 TPS (VISA, 2019); comparatively PayPal averaged 241 TPS in 2017 (PayPal, 2018).
- Credits in a state sponsored system - millions of citizens using services like ID verification, licensing, or voting (Casino et al., 2019).
- Tokenised value exchange - electricity credits moving from solar generation to national grid: smart grids (Pop et al., 2018) or managed between electric cars and a grid.
- Micro subscription services - paying for video content on a per-second basis or per-share on social media (Chakravorty & Rong, 2017).
- Internet of Things (IoT) in general is poised to require orders of magnitude more data-exchange (Christidis & Devetsikiotis, 2016).

Methods that fork the software only to adjust parameters like block-size and block-interval are only a small step towards high-load protocols (Croman et al., 2016). In order for scaling to be achieved, well-defined metrics and measures of performance are required.

Scalability Linked to Performance

Consensus in blockchain systems serves as the backbone for decentralisation, operationalised through message-passing between computing nodes as shown in Figure 2.3. The metrics used for evaluating the efficacy of consensus algorithms are consequently derivative of this message passing. These metrics include transaction throughput, time-to-finality, and network

16. In practice blocks are mined on average every 576 seconds, just under ten minutes, from 2020–2022.

latency, which are further elaborated in terms of message-passing complexity, storage requirements, and fault tolerance (Dinh et al., 2017; Croman et al., 2016; Zamani, Movahedi, & Raykova, 2018).

The utility of these metrics is not static but is modulated by the specifics of the consensus algorithm. Recent research combines classical algorithms such as Paxos and BFT-SMaRt with novel protocols, like Bitcoin's emergent consensus, in pursuit of optimised, scalable solutions (Lamport, 1998; Bessani, Sousa, & Alchieri, 2014). Within this context, scalability is often subsumed under broader performance metrics (Dinh et al., 2017), and performance itself is dissected into categories such as the complexities of the consensus and bootstrap protocols, as well as storage (Zamani et al., 2018).

The analysis in Section 2.4.1 contains a text mapping (Figure 2.10) that places *performance* and *value* coupled together. The term value also refers to *evaluate* and *measure* and their derivatives, thus, placing emphasis on metrics associated with performance. Therefore, from performance the link is made to scalability in the trilemma.

Blockchain performance depends on scaling which in turn depends on the consensus algorithm. As the number of nodes in the network grow, so does the message passing described above and while message size may be static, cumulative storage requirements also grow. This is the crux of the problem when scaling blockchain networks. Many solutions are proposed and are discussed presently.

Improving Performance

The current state of proposed improvements can be split into four categories: Committee-based approaches, Sharding-based approaches, Second-layer approaches, and Rollups.

Committee Based Approaches

A committee represents a subset of the nodes that are available to participate in consensus. This reduces communication overhead in the network as there are fewer nodes that must exchange information about the state. The risk of using a committee is that a subset of nodes may not represent the majority and are easier to subvert.

An early attempt at using committees is by Decker, Seidel, and Wattenhofer (2016). Called PeerCensus, it is built on Bitcoin and uses a PBFT approach to manage identities. Hybrid Consensus (Pass & Shi, 2017) periodically elects a committee consisting of recently online participants and reaches agreement through PBFT. Alogrand (Gilad, Hemo, Micali, Vlachos, & Zeldovich, 2017) selects a committee from the nodes using a verifiable random function. The committee then decides on the next block. Experimental results show 125x Bitcoin's throughput. The RapidChain protocol by Zamani et al. (2018) uses a committee sampled from all nodes to agree on the updated ledger state with sub-linear communication. RapidChain

also optimises storage with a sharding approach, discussed next. Some other notable protocols mentioned in the literature include ByzCoin (Kokoris-Kogias et al., 2016), Dfinity (Hanke, Movahedi, & Williams, 2018), and Solida (Abraham, Malkhi, Nayak, Ren, & Spiegelman, 2016).

`Sharding Based Approaches`

Sharding takes it name from replicated hardware: A System for Highly Available Replicated Data (SHARD), but simply splits a data-table horizontally to allow for a smaller index size. Sharding leads to a hierarchical, and perhaps less decentralised, structure, seen in myriad natural and social systems. Google's Spanner (Corbett et al., 2012) is a database with automatic fail-over and re-sharding and load rebalancing. Spanner maintains 5 geographically separated replicas; BigTable (also by Google) also uses sharding.

Elastico (Luu et al., 2016) is the first candidate for a secure sharding protocol with the presence of Byzantine adversaries; scaling up to 1600 nodes. Rapidchain (above) claims to achieve 7300 TPS and other good metrics via full sharding (Zamani et al., 2018). OmniLedger (Kokoris-Kogias et al., 2018) claim that throughput can scale linearly with the number of validators achieving VISA-level status using sharding. OmniLedger employs the ByzCoinX consensus algorithm (Kokoris-Kogias et al., 2016). RSCoin (Danezis & Meiklejohn, 2016) is an older project based on two-phase commit with sharding, this cryptocurrency framework decouples the generation of the monetary supply from the maintenance of the transaction ledger. RSCoin is designed with the intention of cooperating with a central bank; and is therefore not decentralised. Ethereum's PoS protocol called Casper is based on sharding (Buterin & Griffith, 2017).

A few other optimisations of storage include using Schnorr signatures to combine private keys into single keys (signature aggregation) (Schnorr, 1991), Merkelised Abstract Syntax Trees, and Segregated Witness (Antonopoulos, 2017). Segregated Witness (Segwit) is a good short-term improvement both in terms of performance and governance.

`Second Layer Approaches`

Changes to a protocol codebase involve updates being pushed out to all the nodes. If the community agrees to the update this is a soft fork, however, if there are different views about the direction of the project some may accept the update and some may not, hard-forking the project. A second-layer approach is a bolt-on solution offering added functionality in the form of a second project.

The Lightning Network has much promise for increasing throughput outside the main-chain. The Lightning Network represents an off-chain scaling solution for Bitcoin because the functionality occurs in a separate network with only the endpoints connecting to the Bitcoin blockchain. All the transacting happens in the middle between lightning nodes that don't need to write to the blockchain. Developed independently by Decker and Wattenhofer

(2015) and Poon and Dryja (2016), it uses micropayment channels positioned between the blockchain (settlement layer) and the payment channels. Similar schemes are being developed for Ethereum for payments (Raiden Network, 2019) and contracts (Poon & Buterin, 2017).

Rollups

Scaling layer two solutions by rollups aim to batch transactions off the main chain and publish either a proof they are executed on the main chain or a compressed version of the transaction bundle. The first approach is a zero-knowledge (ZK) approach, which is to provide cryptographic proofs that verify the legitimacy of transactions without revealing their underlying data (Garoffolo, Kaidalov, & Oliynykov, 2020). The second approach is to batch the transactions into a single or a few blocks and then submit these to the main chain, thereby reducing the number of individual transactions that need to be processed and stored on the main chain (Thibault, Sarry, & Hafid, 2022). Both approaches serve to improve scalability, albeit through different mechanisms, and each has its own trade-offs with regard to computational overhead and data privacy.

2.4.3 Security

Third part of the trilemma in Figure 2.8 is security which must be prioritised in design and operation. Compromises to security could lead to loss of money, information exposure, privacy leaks, and any number of non-negotiable issues from a user standpoint. When dealing with decentralised systems that store data in this manner security must be robust to inspire trust from the users. To present a complete and broad review, some of the main security issues are mentioned including cryptography, privacy, system design, and mining.

Cryptography

Contrary to popular belief there is no standardised encryption in the Bitcoin protocol. As a decentralised system of exchange, there is no need for encryption. All the transactions are stored in the blockchain and accessible to everyone. Access to the private keys controlling addresses is maintained solely through personal security of the user. Funds cannot be stolen by hacking because Elliptic Curve Cryptography (ECC) keeps keys safe from cryptanalysis via the difficulty in solving the discrete logarithm problem. The standard curve `secp256k1` was chosen by the designer of the Bitcoin protocol. This is a choice unique to Bitcoin, as the more common `secp256r1` is used in Transport Layer Security (TLS) for web browsing and email. ECC is chosen because it provides the same level of relative security with smaller keys compared to RSA.

Digital signatures allow a blockchain user to transfer ownership of a token by verifying

they have the authority to do so. Additionally this allows the recipient to verify the message has not been altered at any time. Every transaction on a blockchain is digitally signed by the parties involved; this means they have used their private key to attest to the transaction. A benefit of coupling a cryptographic-key/ID system with a blockchain is that you can prove to someone that you did *not* do something. For example, with PoW the blockchain can prove you do some computation, include everyone's input, and do not censor people from participating. Using just cryptography you can prove to someone that you did something like sign a message, but cannot prove the inverse.

Blockchains are only as good as their weakest link and subject to future threats of various kinds, the known unknowns including: quantum computing (Li, Loucks, Zhai, & Zhong, 2018), or other mathematical breakthroughs (Garfinkel, 2018). While possible future advances in computing should not deter blockchain researchers and developers, cryptography, as the name suggests, plays an important role in securing personal data, and privacy.

Privacy

In the realm of blockchain technology, privacy remains a topic of paramount concern. Standard blockchain implementations do not inherently offer comprehensive privacy solutions and are instead characterised as pseudonymous (Narayanan et al., 2016). The pseudonymous nature of such systems can often be misleading, as users tend to underestimate the multitude of techniques available for de-anonymising participants through chain analysis methods.

To address these privacy concerns, zero-knowledge proofs (ZKPs) emerge as a critical cryptographic technique. ZKPs permit a prover to establish the validity of a statement without revealing any information beyond the veracity of the statement itself (Garfinkel, 2018). One salient example is Zcash, which utilises ZKPs to facilitate transactions that do not disclose the sender, the recipient, or the transaction amount (Ben-Sasson et al., 2014). However, the employment of ZKPs introduces computational overhead and necessitates the use of cryptographic protocols that can be complex to audit.

Another illustrative case is Monero, a privacy-centric cryptocurrency that employs a variety of techniques, such as ring-signatures, ring confidential transactions, and stealth addresses, in addition to i2p routing to enhance transactional privacy (Conti, Sandeep, Lal, & Ruj, 2018). Each of these techniques represents an individual protocol, adding multiple layers of complexity to the system.

Privacy protocols are also being expanded into generalised ZK computation that can improve efficiency and overall privacy for blockchains. In a conventional EVM (Section 2.3.3), smart contract execution and transactions are transparent and publicly verifiable, which can compromise privacy. In contrast, zk-EVM aims to maintain the benefits of public verifiabil-

ity while allowing for private transactions. In the zk-EVM architecture, ZKPs are utilised to validate the correctness of transactions without revealing the underlying data (Capko, Vukmirović, & Nedić, 2022). For instance, a user can prove that they have sufficient funds to complete a transaction without disclosing the exact amount in their wallet. Similarly, the integrity of smart contract execution can be verified without revealing the inputs and outputs.

Though zk-EVM presents significant advantages concerning privacy, it also comes with computational and complexity costs. ZKPs often require more computational power to generate and verify, which could impact the overall performance of the Ethereum network if not optimised efficiently. The integration of zk-EVM in existing blockchain systems represents a step toward reconciling the inherent transparency of blockchains with the increasing demand for privacy and confidentiality (A. M. Pinto, 2020). It is an area of ongoing research and development, as solutions are sought to minimise the associated computational overhead while preserving robust security guarantees.

System Design

System design serves as a cornerstone for ensuring security, beginning with the fault tolerance mechanisms inherent to the chosen consensus algorithm. For instance, a blockchain operating on a proof-of-work protocol can only withstand malicious activity from up to 50% of its nodes, or even fewer under specific conditions. Conversely, a consensus algorithm like RAFT can tolerate up to half of its nodes being faulty but is not equipped to handle malicious nodes. These distinctions are systematically categorised in Table 5.3 and show that inherent differences exist that must be considered at the design phase.

The limitations of the consensus algorithm, however, represent merely one dimension of the system's overall security. Poorly implemented smart contracts or inadequate key management could also result in significant vulnerabilities from the user's standpoint. Therefore, formal verification of both protocols and smart contracts becomes indispensable for a robust system design (Casino et al., 2019). Other measures, such as social key recovery methods–whereby a group of trusted individuals can collaborate to restore or create a new master key–are also gaining traction. This method is already employed in platforms like WeChat, uPort, and Sovrin (Dunphy & Petitcolas, 2018). Additionally, features like time-delayed withdrawals and multi-signature authorisation are increasingly being integrated into contemporary blockchain products.

Mining

PoW blockchains face two primary technical vulnerabilities: 51% attacks and strategic mining. A 51% attack becomes possible when a single entity controls over half the hashpower in the network, thereby gaining disproportionate influence over the blockchain's consensus mechanism (Narayanan & Clark, 2017). In such a situation, the attacker can potentially censor transactions by selectively omitting blocks that contain addresses or criteria they wish to suppress.

However, controlling 51% of the hashpower is not a strict requirement for launching an attack. Even with a large but sub-majority hashpower, an adversary could execute a double-spend attack. Eyal and Sirer (2014) elucidate a strategy where a coalition of miners could withhold their blocks, sacrificing immediate rewards to increase long-term gains by eliminating competition with honest miners. This highlights that the mining sector remains an ongoing research focus, particularly as the popularity and adoption of Bitcoin and similar technologies increase.

In conclusion, the multifaceted security considerations intrinsic to blockchain design are not static; they will continue to evolve in tandem with the blockchain's growing user base and complex socio-technical interactions. Given the blockchain trilemma, security remains paramount; any users affected by inadequate design, malicious actors, or broader social dynamics will inevitably seek alternative platforms.

2.5 Research Questions

The research motivation begins with the study of blockchains, narrows to the factors of the blockchain trilemma, and splits into two approaches, each leading to a research question. This is depicted in Figure 2.11.

The blockchain trilemma in Section 2.4 has three areas: decentralisation, scaling, and security. The inherently open nature of participation in decentralised systems necessitates prioritising security. Given that there is no mechanism for blacklisting participants, and the lack of a centralised enforcing authority, the system relies heavily on its inherent security measures. Any deviation from secure operation can lead to an exodus of users, as there are no overarching centralised structures to retain them such as big tech monopolies or state mandated use. Hence, security becomes a non-negotiable aspect of blockchain systems, effectively reducing the trilemma to a dilemma focused on scalability and decentralisation.

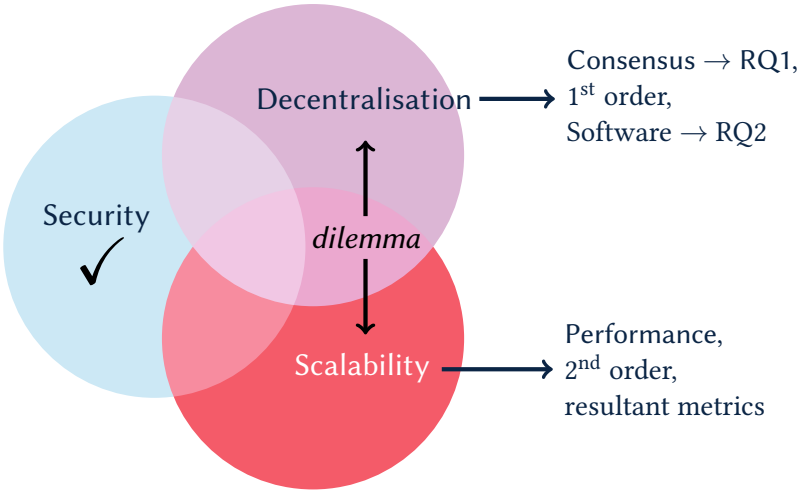


FIGURE 2.11: Reducing the blockchain trilemma to a dilemma places scalability and decentralisation as distinct concepts. Considering decentralisation as the foundation related to the consensus method leads to two branches for research: one investigating consensus methods and the second investigating software.

Scalability, in this framework, is the capacity of the system to adapt to an increasing user base, thereby facilitating greater transaction throughput and data storage. The primary measure for evaluating such capability is TPS. Though other performance metrics are pertinent (Section 2.4.2), they assume secondary importance and are mainly relevant when considered in relation to TPS. The dilemma hence transitions into a more specified form: scalability becomes tantamount to performance.

Simultaneously, decentralisation manifests through the use of consensus methods. Centralised networks only require crash fault tolerance as there are no malicious actors to consider. Whereas networks most distributed in nature utilise BFT algorithms to achieve consensus, as illustrated by the cluster analysis in Figure 2.10. Thus, the degree of a network's decentralisation is intrinsically linked to its consensus algorithm.

Building upon this premise, the blockchain's primary design consideration is the consensus method that enables its existence as a decentralised system. Performance and scalability, then, emerge as secondary concerns. This ordering of design priorities provides the perspective for future research directions. Firstly, the consensus method becomes the focus, given its role as the algorithmic backbone for maintaining the blockchain. Secondly, acknowledging that a blockchain network is fundamentally a software system prompts questions about overall health and success.

These two distinct research directions each lead to questions. The first research question is motivated from the study of blockchains and the relationship of factors in the blockchain

trilemma.

Research Question 1

What are the factors that influence blockchain consensus?

The second question arises from literature on software health. With regards to performance being a consequence of the consensus method, is there a way to gauge success of the myriad types of consensus? Success is closely related to health and this is the topic of review next, in Chapter 3.

2.6 Conclusion

Originating from the domain of distributed computing, blockchain systems are initially conceptualised to address the issue of digital cash. The blockchain trilemma posits that a blockchain design must compromise on at least one of three attributes: decentralisation, scalability, and security. Given the imperative of security in such systems, the trilemma narrows to a dilemma, centering on consensus methods as the primary design consideration and scalability as a consequential concern. In the context of this thesis, the identified knowledge gap lies in understanding the landscape of blockchain consensus methods. This leads to the formulation of the first research question: What are the factors that influence blockchain consensus? The subsequent pathway focusing on the health of blockchain systems is next, in Chapter 3.

Chapter 3

Software Health Literature Review

> *Given enough eyeballs, all bugs are shallow.*
>
> Eric S. Raymond
> *The Cathedral and the Bazaar*
> 1999

Chapter 2 Reviews the State of blockchain literature from the introduction of Bitcoin in 2009 to the issues posed in the blockchain trilemma, mainly that a blockchain experiences limits in scaling while remaining decentralised. This chapter defines the open source environment that decentralised blockchains belong to with regards to studying health. Open source software is introduced in Section 3.1, and software health is reviewed and defined in Section 3.2. Models of software health are overviewed in Section 3.2.3. The three characteristics of healthy software are detailed along with their metrics in Section 3.3–Sustainability, Section 3.4–Robustness, and Section 3.5–Niche Fit. The applicability of health methods to blockchain is discussed in Section 3.6. To finish, research questions are posed in Section 3.7, preceding a conclusion in Section 3.8.

3.1 Open Source Software

Open source software begins with the free software movement in 1983 when Richard Stallman announces he is going to write an alternative to UNIX, GNU, and allow users to copy, improve, and redistribute the code (Stallman, 2009). The Free Software Foundation (FSF) follows in 1985 to manage the project and advocate for users that want to fix their own bugs, distribute updates, and *scratch an itch*[1] left from proprietary software[2] to add functionality.

The free software movement in the 80s resulted in the GNU General Public Licence (GPL), the first licence allowing users to modify the source code with the restriction that derivatives include the same licence (Raymond, 1999). Software utilising the GPL is signalling that freedom of its users comes first. "'Free software' is a matter of liberty, not price. To understand the concept, you should think of 'free' as in 'free speech,' not as in 'free beer"' (The Free Software Foundation, 1989).[3] This category is now known as Free and Open Source Software (FOSS) or free/libre open source software (FLOSS). This work uses FOSS as inclusive of FLOSS. Figure 3.1 shows some of the key events in the timeline of FOSS to OSS and presently into the blockchain era.

By the end of the 80s, with people and companies collaborating, a few prominent cases of dual licensing emerge such that a free open source version is released to inspire users to purchase the proprietary version that is packaged with enhanced features (Gonzalez-Barahona, 2021).

1. "Every good work of software starts by scratching a developer's personal itch" by Eric Raymond. See http://www.redhat.com/redhat/cathedral-bazaar/.
2. Proprietary Software can be said to have begun in 1969 when IBM unbundled software from its hardware and began selling it independently.
3. See https://www.gnu.org/philosophy/free-sw.html.

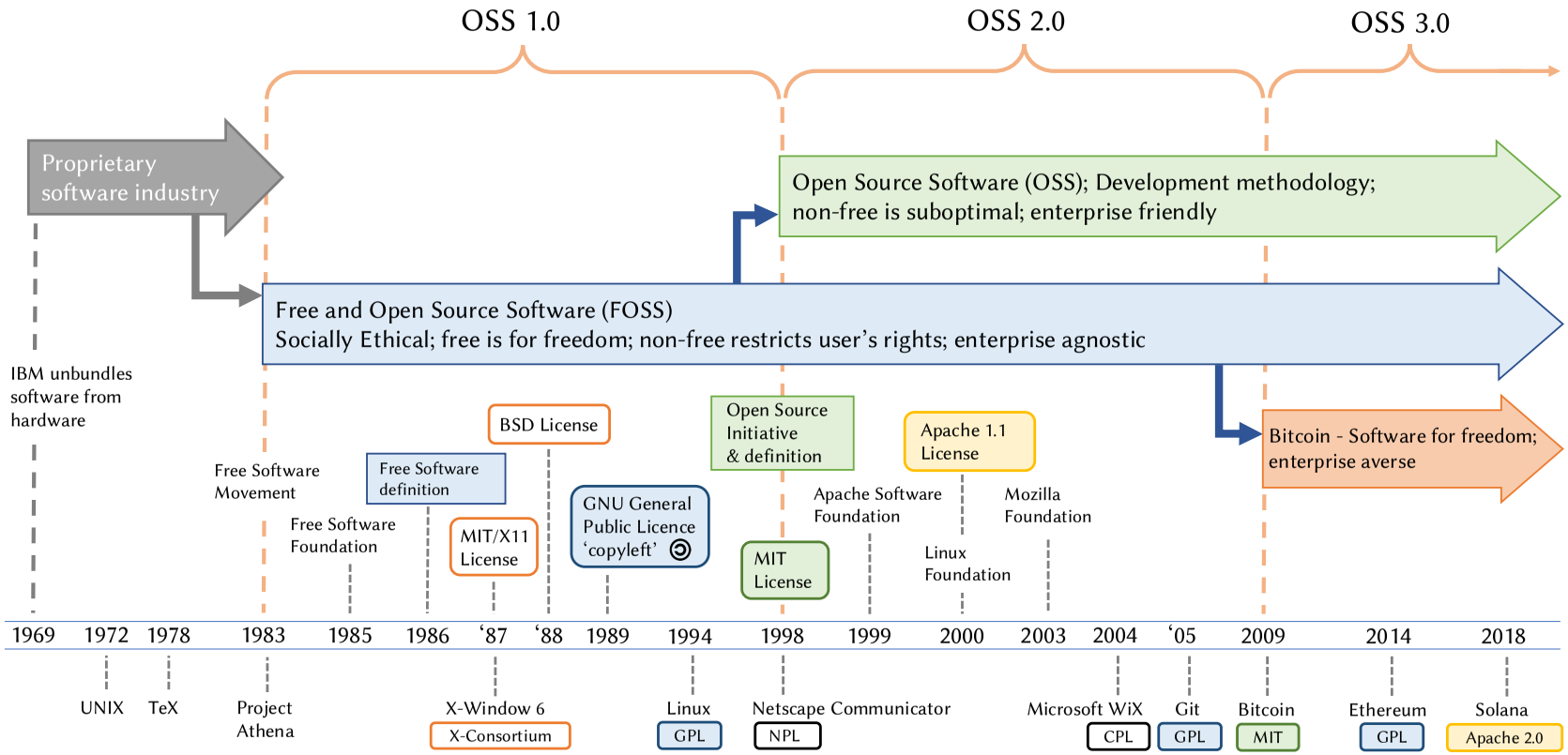


FIGURE 3.1: Key events in the history of open source software. The third era begins in 2009 when Bitcoin software is released under the MIT/X11 licence. Selected software releases are along the bottom with their associated Open Source licence, if any. **BSD** is Berkeley Software Distribution; **GPL** is GNU Public Licence; **NPL** is Netscape Public Licence; **CPL** is Common Public Licence.

A second generation of open source emerges in 1998 when Netscape chooses to release its web browser Netscape Communicator as FOSS (Gonzalez-Barahona, 2021) forming the Open Source Initiative (OSI) to steward the project and develop the licence.[4] Netscape is an important early internet company and its browser competed with Microsoft Internet Explorer.[5] Corporations at this time are not ignorant of FOSS, rather they have a dismissive attitude: "Who would trust a mission-critical application to software written by a bunch of long-haired anarchists?" (Anthes, 2016). Netscape's move to open source their browser is radical and so to prepare enterprises for the news they opted to use the term 'open source' to be more business friendly than the ambiguous term 'free' (Fitzgerald, 2006; Gonzalez-Barahona, 2021).

Companies in the OSS 2.0 era leaned in to the reciprocal nature of the licences to strengthen their brand, endear the OSS community that is traditionally adverse to their proprietary models, and to embrace positive network externalities (Fitzgerald, 2006). Companies today rely heavily on OSS code – in 2016 there are 84 OSS components per commercial application which rises to 528 components by 2020 (Crisman, Logan, & Bansal, 2021). Microsoft first joined OSS 2.0 in 2004 when it released its WiX[6] under the OSI approved Common Public Licence (CPL). By 2016 Microsoft joins the Linux Foundation, as Lindman (2021) says, "this redefinition strategy can be seen as successful: OSS became mainstream." In 2015, 78% of enterprises surveyed run at least part of their business on OSS (Yamashita, Kamei, McIntosh, Hassan, & Ubayashi, 2016) which is nearly double from 42% only five years earlier. Some OSS components like OpenSSL and Apache are so prominent—OpenSSL is used in 84% of the top 1 million websites (Nemec, Klinec, Svenda, Sekan, & Matyas, 2017); Apache and NGINX account for 65% of web servers[7]—that commentators have suggested mainstream has become mandatory, leaving purely proprietary software in a niche market (Anthes, 2016). "New projects today are open source by default, unless there are good reasons why they shouldn't be," Anthes says. "That's a complete switch from the proprietary mind-set of earlier days" (Anthes, 2016).

3.1.1 Definitions of Open Source Software

Each era has its representative definition of what open source software means. Stallman and the The Free Software Foundation (1989) write the **Free Software definition** in 1986[8] as the freedom to:

4. This is also motivated by Eric Raymond publishing *The Cathedral & the Bazaar* (1999) seven months earlier.
5. Another pioneer is NCSA Mosaic developed at the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana–Champaign. Mosaic made available source code for the UNIX version, but does not release the software as OSS (Wolfe, 1994).
6. Windows Installer XML, see https://www.computerworld.com/article/2564230/microsoft-goes-open-source-with-wix-tool.html.
7. See https://www.stackscale.com/blog/top-web-servers/.
8. First appeared in 1986, see https://www.gnu.org/bulletins/bull1.txt.

1. Run the program as you wish for any purpose.
2. Study how the program works, and change it so it does your computing as you wish.
3. Redistribute copies so you can help others.
4. Distribute copies of your modified versions to others.

Notably, points two and four require access to the source code, while points one, two, and the combination of three and four represent three freedoms that the definition requires for the software to be considered 'free software.' *The Cathedral and the Bazaar* (Raymond, 1999) highlights some issues and leads to a revision of the FSF's definition. Companies writing enterprise software recognised the need for modified licensing such that future versions could be proprietary.

This means starting with FOSS, modifying, and relicensing to prevent unwanted distribution. The OSI publishes their own ten point definition (Open Source Initiative, 2023) based on the Debian Free Software Guidelines developed in 1997.[9] This is authored by Perens at Netscape during Communicator's code release.

The OSI definition of Open Source:

1. Free redistribution: the software must be able to be freely given away or sold.
2. Source code must be included or freely obtained.
3. Derivative works: redistribution of modifications must be allowed.
4. Integrity of the author's source code - Licences may require modifications to be redistributed only as patches.
5. No discrimination of individuals or groups: no one can be left out.
6. No discrimination of initiative areas: Business users cannot be excluded.
7. Licence distribution - The same rights should apply to everyone who receives the program.
8. The licence should not be specific of a product: the program cannot be licensed only as part of a larger distribution.
9. The licence should not restrict other software: the licence cannot oblige that any other software that is distributed with the open software must also be open source.
10. The licence must be technologically neutral - Acceptance of the licence by mouse click access or otherwise specific to the software medium should not be required.

Although more verbose, the OSI definition is inclusive of the meaning of the FSF definition such that all free software is open source but not all open source software is free. The main point of difference in the definitions is in point nine where it specifies that modified versions cannot enforce an inherited licence. Thus, companies are free to base software on OSS com-

9. https://www.debian.org/social_contract#guidelines

ponents, turn around and licence to their customers with a proprietary agreement or more restrictive licence.

3.1.2 GitHub

Git, which is created by Linus Torvalds in 2005 to manage the Linux kernel source code (Loeliger & McCullough, 2012), is the inspiration behind GitHub.[10] Git remains a cornerstone in workflow of software development. Central to its design is the concept of a distributed version control. Unlike many of its predecessors, which rely on a single centralised repository, Git adopts a different philosophy. Each clone of a Git repository stands as a full-fledged repository on its own, with comprehensive history and tracking capabilities. This means that developers have the entire history of the project on their local machines rather than checking out a snapshot (Spinellis, 2012). This distributed approach is not merely a technical distinction but a revolutionary shift. It empowers contributors to work independently, at their own pace, in their own time.

Git is responsible for normalising the software practices of branching and merging. Branching is an integral part of workflow allowing developers to swiftly create, switch to, and work on separate branches stemming from the same code base. Merging happens when a branch is ready to be assimilated back into main branch. This agility in handling branches fosters an environment where parallel lines of development can converge, enhancing the collaborative potential of teams.

Several factors underscore the widespread adoption of Git. Its distributed framework allows each contributor a full repository, facilitating offline work and offering a protective shield against singular points of failure (Favell, 2020). This decentralised approach is complemented by Git's natural branching and merging, streamlining collaboration for both individual contributors and teams, even amidst concurrent developmental trajectories. The ascent of Git aligns with a cultural transition in software development (Figure 3.1), as the industry increasingly leans towards open source collaboration, recognising in Git a platform that embodies this collaborative spirit.

The foundation of GitHub is firmly anchored in the OSS ethos, advancing the philosophy of collaborative development and transparent knowledge sharing (Lima, Rossi, & Musolesi, 2014). The concept of forking is a defining feature offered by GitHub, which has contributed to its success. Forking refers to the "behavior of copying an existing project's code base" (Negoita, Vial, Shaikh, & Labbe, 2019, p.2) as shown in Figure 3.2. This allows individuals to make changes independently of the original codebase. The origins of this term

10. https://github.com/

trace back to the early days of software development where a process creates a copy of itself (Robles & González-Barahona, 2012), symbolising the branching out of a project into different developmental trajectories. A pull request (PR) is a request from a contributor (not necessarily known to the project developers) to pull the new branch back into the main branch (Figure 3.2). Through its intuitive interface and community-driven initiatives, GitHub has significantly fostered the culture of forking and outshone other pull-based competitors such as Bitbucket (Gousios, Storey, & Bacchelli, 2016). The platform not only simplifies the process but also propagates an ethos of collaboration.

Some call GitHub a social coding platform (Vasilescu et al., 2016; Lima et al., 2014), while others describe it as a paradox of centralised decentralisation (Metz, 2015). Either way, GitHub represents the largest OSS community hosting over 254 million repositories by 73 million developers (GitHub, 2021; Gousios et al., 2016; Hu, Zhang, Bai, Yu, & Yang, 2016). Data for public blockchain projects are readily available for collection and analysis from GitHub through the web interface, programmatically through the Application Programming Interface (API), and in raw archival form from the GitHub Archive.

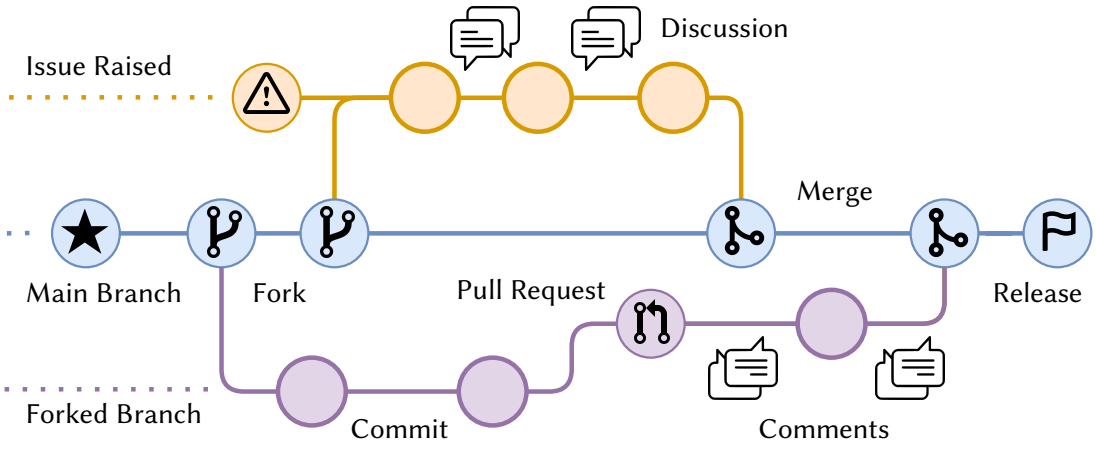


FIGURE 3.2: The GitHub project lifecycle from instantiation at the main branch to forking, submitting a pull-request, commenting on changes and issues, finally to merging back into the main branch. All of these event types are tracked.

Figure 3.2 show a project lifecycle that typically begins as a personal endeavour or the initiative of a small group. As they garner visibility, they draw potential contributors who can fork the project, contribute, and eventually reintegrate their changes into the primary codebase. Contributors can propose changes to the original repository through a pull request, bridging the gap between individual endeavours and collective evolution. Through the entire process there is intuitive functionality to comment on changes and revisions in threaded conversations.

Returning to Linus Torvalds' motivation for creating Git, at the time Linux kernel maintainers are using BitKeeper which is closed-source and proprietary. BitKeeper revoked the free-to-use licence when someone started to reverse-engineer it with a view to creating an open source replacement tool, and Linus sees no choice except to create one from scratch which he names Git (Z. Brown, 2018).[11] Since inception, Git has a free and permissive license which encourages widespread use.

3.1.3 OSS Licensing

Defining what open source means and what users understand by open source is related to the licence that is distributed with the software. The legal default for a creative work including code, art, writing, music, and video falls under copyright law giving exclusive rights to the creator or group. OSS is unique in the sense that the primary motivation for licensing is to enable others to distribute, modify, and copy the source code (Laurent, 2004). The licence specifically states what activities are and are not encouraged and permitted.

According to GitHub's Terms of Service[12] users that create a public repository give the right to other users to fork that repository. There is the option for private repositories but the default option allows forking. This public-by-default attitude contributes to the permissive nature of open source. The Terms of Service alone, in this case GitHub's, do not legally allow others to copy and modify code and so a licence is required.

GitHub lists 36 different licences, seven of which are part of the GNU GPL family.[13] New licences can be requested to be included in the set as needs arise. Recent additions aim for brevity and simplicity such as Creative Commons Zero (CC0) and the Unlicence. The OSI approves licences before committing them to their official listing and as of 2022 there are 111 approved licences, including the GPL, Apache, and MIT families.[14]

The three most used licences within GitHub are MIT, Apache 2.0, and GPLv3 (GitHub, 2023). The origins of the MIT licence are with Project Athena in 1983, a partnership between DEC, IBM, and MIT to develop on-campus computing. From this project comes the X Window system and Kerberos which are licensed with this X Consortium licence beginning with version 6, or X Window 6 in Figure 3.1.

The modern MIT licence begins with the OSI in 1998 and allows for software dependen-

11. Torvalds says "The in-joke was that I name all my projects after myself, and this one was named 'Git'. Git is British slang for 'stupid person'," (Favell, 2020).
12. Available at https://docs.github.com/en/site-policy/github-terms/github-terms-of-service
13. As of writing, see https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/licensing-a-repository#searching-github-by-license-type.
14. https://opensource.org/licenses/alphabetical

cies (Laurent, 2004). Its also called the X-11 licence after the eleventh version of the X Window system. The MIT licence is a form of copyright licence that grants permission to use, copy, modify, merge, publish, distribute, sublicence, and/or sell software, subject to certain conditions. While modifications are allowed, there is debate about whether they are compatible with the original licence terms and whether they can still be considered "free" in the sense of open source software (Laurent, 2004).

It is similar to the Berkeley Software Distribution (BSD) licence first used a year later in 1988 to allow distribution of Berkeley's networking code separate from AT&T's expensive licensed version (McKusick, 1999). BSD refers to the software distribution package that started in 1977 as graduate students at Berkeley modified UNIX, whereas the BSD licence is not necessary until the need to freely (as in beer) distribute the code in 1988.

Apache 2.0 licensing is prevalent in the business community (Kripalani, 2017). It requires a disclosure statement to be made about any major changes to the original code that take place. The code itself does not need to be revealed allowing for commercial projects to succeed, including the option of patenting the modified version. As patents place restrictions on future liberties of developers and end users, this proprietary leaning stance disagrees with the intent of the free software movement. The counter position is the GPLv3 licence which restricts future inclusion in closed source projects. Each project that is licensed with GPLv3 must inherit the parent licence as per Section 5 - Conveying Modified Source Versions: part (c):

> You must license the entire work, as a whole, under this Licence to anyone who comes into possession of a copy. This Licence will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This Licence gives no permission to licence the work in any other way, but it does not invalidate such permission if you have separately received it. (Open Source Initiative, 2007)

The GNU GPL is also referred to as the copyleft licence (🄯) meaning that when distributing you cannot change the licence which could potentially restrict the freedoms of future users and developers.

3.1.4 Blockchain Software Licensing

A large proportion of blockchain projects are open sourced with the present study finding that 69% of the top 600 blockchain projects, listed on CoinMarketCap[15] as of March 2022, having publicly visible code repositories on GitHub. Bitcoin, as the first blockchain project, lists

15. https://coinmarketcap.com

MIT/X11 as the software licence in Figure 3.3. The MIT/X11 licence begins in 1987 (Figure 3.1), and is now known as the MIT licence which allows for commercial and licence modification for derivatives.

```
BitCoin v0.01 ALPHA

Copyright (c) 2009 Satoshi Nakamoto
Distributed under the MIT/X11 software license, see the accompanying file
license.txt or http://www.opensource.org/licenses/mit-license.php.  This
product includes software developed by the OpenSSL Project for use in the
OpenSSL Toolkit (http://www.openssl.org/).  This product includes crypto-
graphic software written by Eric Young (eay@cryptsoft.com).
```

FIGURE 3.3: Bitcoin software version 0.01 `readme.txt` showing the MIT/X11 licensing. Originally published on SourceForge (Satoshi Nakamoto Institute, 2023).

Ethereum's source code uses a GPL Lesser v3 licence[16] shown in Figure 3.4. The Lesser designation is not as restrictive as the base GPL licence but still requires modifications to carry the same licence. The risk with this copyleft licensing by the Ethereum Foundation is that the restrictive nature of the derivative versions' licence can infect future software leaving developers wary of incorporating Ethereum modules.

Whereas OSS 2.0 companies relied on the contributions of OSS developers to create data silos that generate corporate profits from user activity and data, the next movement in open source is committed to strengthening user data while remaining neutral to corporate influence. This third era, known as OSS 3.0, is firmly grounded in decentralisation. Figure 3.1 illustrates some of the key events leading to this development. There is an overlap with Web 3.0, which focuses on user-owned data and user-verified data sharing experiences. For Web 3.0 to succeed, OSS 3.0 must promote collaboration and vigorously protect individual rights and freedoms. According to Allison Randal, president of the OSI, by 2010, "the tide of opinion had flowed overwhelmingly from proprietary software to OSS" (Anthes, 2016). Over the intervening decade, the tide receded on OSS 2.0, revealing an abundance of abuse of user data by monopolistic tech companies eroding the public's trust in corporate technology (Zuboff, 2019). Although not motivated specifically by technology companies' cavalier approach to data, Bitcoin's decentralised ledger quickly becomes recognised as a potential means for users to reclaim data.

Even with established permissive licensing, OSS 3.0 is taking it a step further by open sourcing hardware components for infrastructure in addition to software. Block (previously known as Square) aims to open source components that promote decentralisation, in line with

16. https://github.com/ethereum/go-ethereum/blob/master/COPYING

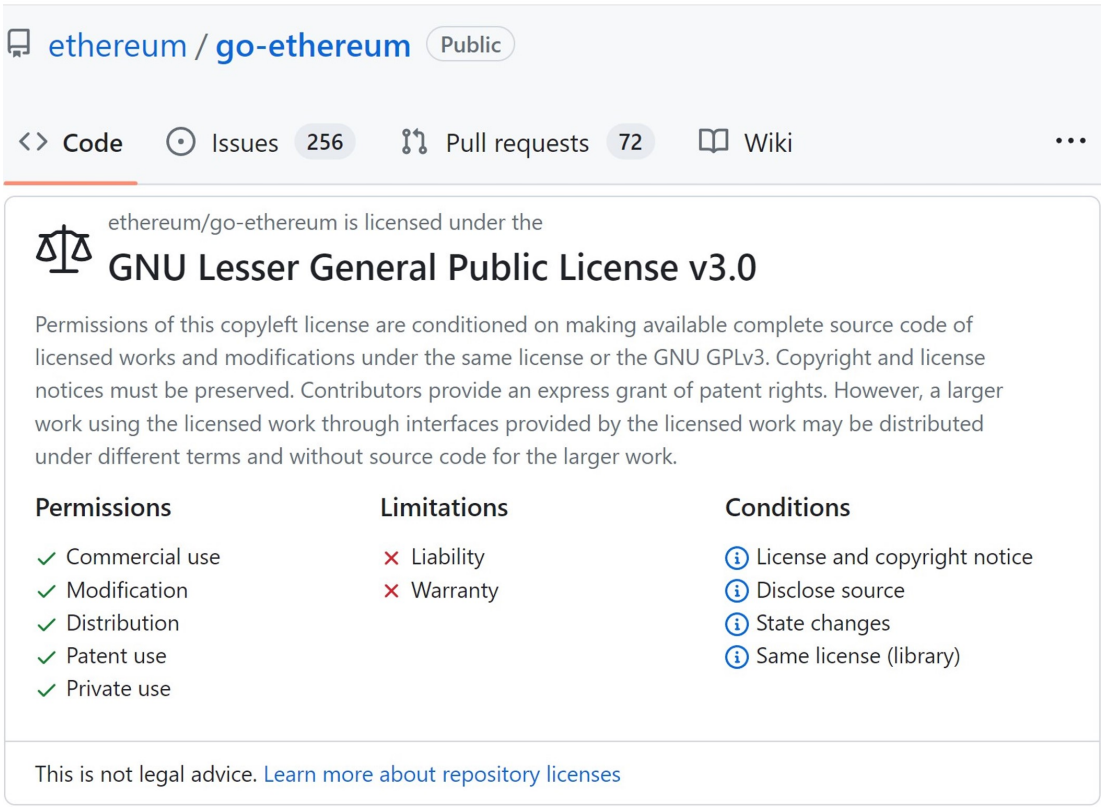


FIGURE 3.4: Source code repository licence for the Go Ethereum client as listed on GitHub with the GNU Lesser General Public Licence v3.0. The conditions state the licence must be maintained in future versions, and any changes stated.

the blockchain trilemma (Section 2.4). An example of this initiative is the Stratum V2 software to help Bitcoin miners collaborate and the associated ASIC mining hardware (Sigalos, 2022). Open sourcing chip architecture such as the RISC-V instruction set used in Google's Titan M2 Security chip can further decentralise sociotechnical systems by removing expensive licensing fees by chip designers that are often subject to export controls and giving citizens the choice to an alternative (Waterman & Asanović, 2017; C. Miller, 2022). In *The Cathedral and the Bazaar*, Raymond (1999) predicts that the future of open source "will increasingly belong to people who leave behind the cathedral and embrace the bazaar." In the software bazaar, each transaction is owned by individual parties, without the need for reporting, committees, or budget taxation. Open source in the bazaar has evolved rapidly, with ground-level innovators able to act quickly and in their own best interest, unencumbered by bureaucratic hurdles. With OSS 3.0, the bazaar is further supported, perhaps allowing it to thrive long before those in the cathedral can come down to participate.

3.2 Software Health

To investigate the issue of software health within the blockchain space, it is appropriate to begin with research into health of OSS generally. In 2007 the first study focussing solely on the concept of health in a software engineering project appears (Wahyudin, Mustofa, Schatten, Biffl, & Tjoa, 2007). Prior to this, the more common project assessment measure is success (Weiss, 2005) as an operationalisation of software health (Goggins et al., 2021).

From 2010 forward, there are a number of studies to analyse and measure software success, health, project sustainability, survivability, and risk. The reason for the growth in number and diversity of studies is the rise of publicly available data metrics. Software built with a version control system in a public arena such as SourceForge[17] or GitHub allows for data to easily be analysed by third-parties. The rise and ease of access through APIs to GitHub or to archival data through GHTorrent[18] and GHArchive[19] makes this an obvious mine to look for trends. In this manner researchers have concentrated on defining metrics with the available data, and although many software metrics exist, there is ambiguity with respect to a definition of health in the open source software context (Goggins et al., 2021).

A software health definition is imperative as it is not well-defined or easily elucidated (Goggins et al., 2021). A top level view of health is "a project's ability to continue to produce quality software" (Goggins et al., 2021; Link & Germonprez, 2018). This definition comes out of prior research into the health of natural ecosystems as applied to business ecosystems, software ecosystems, and also open source software generally. To determine metrics that can be analysed as pertain to health many of the terms in Goggins et al.'s definition need further explanation. The relevant literature is summarised in Table 3.1.

3.2.1 Ecosystem Health as a Metaphor for Software Health

Defining health in the context of software, and further blockchain software, first benefits from a view of health as seen in the life sciences. The health of natural ecosystems and their components, such as soil, water, flora, and fauna, is a pressing concern for the entire biosphere. To understand what constitutes natural ecosystem health, it is useful to draw on metaphors, even those from human medicine (D. J. Rapport, 1989; Hartigh, Visscher, & Tol, 2013). Metaphor has a legitimate place in science as it can stimulate associations between seemingly unrelated phenomena and highlight their structural identity (Rapoport, 1983).

The ecological metaphor is used extensively to relate natural ecosystems to both business

17. https://sourceforge.net/
18. https://ghtorrent.org/
19. https://www.gharchive.org/

ecosystems (Moore, 1993; Iansiti & Levien, 2004; Hartigh et al., 2013), and software ecosystems (Dhungana, Groher, Schludermann, & Biffl, 2010; Chengalur-Smith, Sidorova, & Daniel, 2010; Goeminne & Mens, 2013; Manikas & Hansen, 2013; Jansen, 2014), and allows one to draw parallels on the basis of health. Both natural and software ecosystems are composed of interrelated components, such as species in natural ecosystems and projects in software ecosystems that exist in a competitive environment. Both ecosystems rely on (bio)diversity in order to thrive, and an underlying principle in both is that of adaptation and evolution of the components within the system in order to ensure its survival and continuing success. By learning from the relationships in natural ecosystems, factors can be identified that are crucial to the sustainability and overall health in software.

3.2.2 A Definition of Software Health

A conceptual map illustrating the different ways the idea of health is defined across natural, business, software, and open source ecosystems is presented in Table 3.1. In the context of sociotechnical systems, concept mapping can be a valuable tool for understanding and analysing the complex interactions between social and technical components (Morgan & Guevara, 2008).

The ecosystem health concept map shows the key terms used to describe health in each ecosystem and categorises them into three groups: sustainability, robustness, and niche fit. The terms are identified through a comparative analysis of ecosystem-specific literature, and provide insights into the challenges present when defining health.

Three concepts are synthesised from the literature and serve as the definition of software health.

Definition of Software Health

Health, in the open source software ecosystem context is composed of three factors:

1. Sustainability of day-to-day operations,
2. Robustness to stress, and
3. Niche occupation of the project within the software's local ecosystem.

Before examining each component in turn with its associated metrics, models of software health are brought in to disambiguate them from software success.

3.2.3 Models of Software Health and Success

Health is often conflated with success with each term acting as a proxy for the other. Goggins et al. (2021) survey twenty-two studies on open source community health and find thirteen

TABLE 3.1: Conceptual map illustrating the different ways in which the concept of health is defined across natural, business, software, and open source ecosystems. This map shows the key terms used to describe health in each ecosystem and categorises them into three groups: sustainability, robustness, and niche fit.

Ecosystem	Classifier 1	Classifier 2	Classifier 3	Year
Natural	Productivity	Absence of disease	Diversity	1988[a]
	Sustainability	Integrity; Stress capacity		1989[b]
	Vigour	Resilience	Organisation	'92[c],'98[d]
Business	Growth	Profitability	Stable value-creation	1993[e]
	Productivity	Robustness	Niche creation	2004[f]
	Financial health	Centrality; Visibility	Variety of partners	2013[g]
SECO*	Productivity	Robustness	Niche creation	2010[h]
	Productive	Endure	Variable	2013[i]
OSS†	Liveness of users/devs			2007[j]
	Software development	Long-term		2010[k]
	Vigour	Resilience	AMI⋆	2012[l]
	Sustainability; Maintenance capacity	Resource health	Network health; Process maturity	2014[m]
	Healthy community	Healthy commons		2015[n]
	Community	Code; Resources		2018[o]
	Sustainability	Survivability		2021[p]
Concept	*Sustainability*	*Robustness*	*Niche Fit*	

* SECO is software ecosystem
† OSS is open source software
⋆ AMI is average mutual information
Source literature: [a]Schaeffer, Herricks, and Kerster (1988); [b]D. J. Rapport (1989); [c]Costanza (1992); [d]D. Rapport, Costanza, and McMichael (1998); [e]Moore (1993); [f]Iansiti and Levien (2004); [g]Hartigh et al. (2013); [h]van den Berk, Jansen, and Luinenburg (2010) ; [i]Manikas and Hansen (2013) ; [j]Wahyudin et al. (2007); [k]Chengalur-Smith et al. (2010); [l]Raja and Tretter (2012); [m]Franco-Bedoya, Ameller, Costal, and Franch (2014); [n]Naparat, Cahalane, and Finnegan (2015); [o]Link and Germonprez (2018); [p]Goggins et al. (2021).

of them concentrating on success beginning in 2002 with only four studies concentrating on health beginning in 2014. The remainder focus on sustainability or risk. Take, for example, the concept of activity as a measure for health and success. Negoita et al. (2019) classifies activity as a health metric, Chengalur-Smith et al. (2010) says it's a sustainability metric, and Ghapanchi (2015) says it's a success metric. It is clear that success and health are related, but its not clear to what extent, or what exactly differentiates them other than the researcher's framing. So, how can one measure and gauge the success of a project?

Models of success as an objective measure of teams and their artefacts is valuable to practitioners and academics in Information Systems (IS) seeking to improve efficiency and

output (DeLone & McLean, 1992; Grover, Jeong, & Segars, 1996; Smithson & Hirschheim, 1998). The model by DeLone and McLean proves particularly useful and is applied to e-commerce (Rouibah, Lowry, & Al-Mutairi, 2015), knowledge repositories (Qian & Bock, 2005), user satisfaction (Bharati & Chaudhury, 2006), and software project effectiveness measures (Crowston & Howison, 2011) among others.

The model takes two inputs: system quality and information quality. System quality relates to the day-to-day operations in question, and for software projects equates to activity from the community and developers. Information quality is the software artefact or sub-artefact deriving from commits, debate, review, and ultimately the quality of the software release. The model inputs are then processed by users and the impact on organisations and individuals evaluated. Increasing the resolution, Ghapanchi, Aurum, and Low (2011) suggest project performance is an additional contributor along with quality and activity, although this suggests project performance is an input. The dual inputs of system and information are commensurate with the Naparat et al. (2015) definition of sustainability and health as a healthy community (system) combined with a healthy commons (information).

Using DeLone and McLean's model as inspiration, and cross referencing with the definition of software health from Section 3.2.2 a new model of software health is shown in Figure 3.5. The project level inputs consist of productivity and sustainability, or the regular daily operations of a software project, combined with the robustness of the project adapting the term from survivability of a species. Sustainability is further investigated in Section 3.3. Closing the inputs at this stage, the resulting outputs progress through use and iteration first at an individual level. This leads to growth as part of a successful project although does not alone guarantee success. DeLone and McLean judged success by impact at the organisational level. Impact of a software project can be paralleled by its occupation of a niche in the local landscape. If a project is not impactful enough, it could indicate many players serving the same customer set whereas a high impact project can clearly be delineated in the ecosystem as serving a particular function.

In summary, health is not synonymous with success; although there is a high likelihood for successful projects to also be healthy, a healthy project alone does not guarantee success. Factors such as leadership, strategy, communication, and luck play significant roles in contributing to a project's success. Clear project goals or vision, effective communication within the project, and teamwork or team coordination are often cited as vital components for project success (Nasir & Sahibuddin, 2011; Chow & Cao, 2008; Sudhakar, 2012).

Chow and Cao (2008) identify three critical success factors for software development projects, specifically highlighting delivery strategy, Agile software engineering techniques, and team capability. Furthermore, Sudhakar (2012) conduct a review of 35 critical success factors for software projects and discover that factors such as communication within the project,

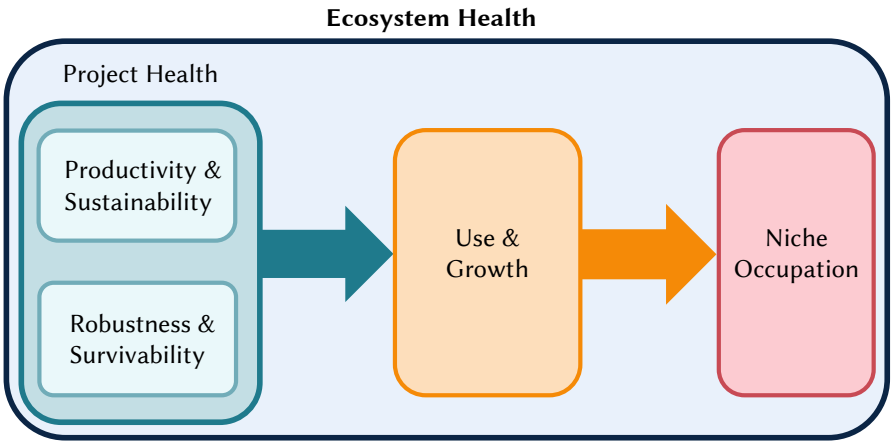


FIGURE 3.5: A model of software health as IS success. This is adapted from DeLone and McLean (1992)'s *A model of success in IS* for software ecosystem health. Software productivity and sustainability represents IS system quality, robustness and survivability represent information quality, and the niche occupation represents the overall organisational impact.

top management support, clear project goals, reliability of output, project planning, teamwork, and project team coordination play crucial roles. Many of these traits are non-technical factors, primarily involving human communication and coordination. Nasir and Sahibuddin (2011) emphasise that non-technical factors dominate in critical success measures over technical ones like skilled staff and familiarisation with development methodologies.

It is worth noting that these traits are less likely to apply to blockchain software, which typically exhibits a more organic, bottom-up, and flat organisational structure. Although this study is not explicitly about success, the relationship between innovation in software and success is conceptually close by. The model seen in Figure 3.5 is steeped in history based on ecosystem health, and the artefact in this research is a framework for project health evaluation. The prime distinction being that project health assumes a lower level perspective and is blind to the broader ecosystem. More on this in Section 3.6.

3.3 Sustainability

From the concept map in Table 3.1 a definition of software health emerges as a combination of sustainability, robustness, and niche occupation. Each of these are explained, along with metrics for measurement, beginning with sustainability.

Sustainability in a natural ecosystem is also referred to as stability (Costanza, 1992), vigour (D. Rapport et al., 1998), and productivity (Schaeffer et al., 1988), and generally refers to the ecosystem's ability to carry out the basic functions necessary for metabolism and growth. In-

dicators that an ecosystem is functioning include: primary productivity – how much growth is occurring, the nutrient base available, the diversity of species present, the amount of instability, disease prevalence, diversity of size spectra, and levels of contaminants (D. J. Rapport, 1989). The list provides a first draft of metrics biologists can track to asses sustainability in a natural ecosystem.

If a natural ecosystem is an interrelated collection of species, a business ecosystem is an interrelated collection of businesses across industries that are both in competition and cooperation with each other (Moore, 1993). Further, the health of that business ecosystem is an aggregate of the stability it needs to be profitable and the stability it needs to grow. Sustainability here is the productivity that comes from general tasks employees undertake to maintain business operations (Iansiti & Levien, 2004). Just as base metabolic resources are required to keep species in an ecosystem in competition, healthy financial resources can sustain a business ecosystem to provide the opportunity for innovation and growth (Hartigh et al., 2013).

Drawing from the definition by Jansen, Finkelstein, and Brinkkemper (2009), a software ecosystem (SECO) is a group of stakeholders "functioning as a unit and interacting with a shared market for software and services." Software ecosystems as a class of business ecosystems often operate through a common technological platform, such as Apple's iOS, and participate in the affiliated markets, such as Apple's App Store.

Sustainability within a SECO is doing enough of the minimum viable activity to run day-to-day operations. When done well this productivity allows a software business to compete and possibly thrive. When done poorly a lack of productivity results in losing market share to a competitor. A productive SECO's outputs include software development activities such as writing code, reviewing, and feature implementation (Manikas & Hansen, 2013). Ensuring the sustainability of the SECO is a complex process that requires significant community effort and resources from the ideation stage to the version release stage (Negoita et al., 2019). In modern software development, a product is no longer viewed as a static entity after its initial release; rather, there is a constant need for user feedback, bug fixing, and iteration, all of which are activities that contribute to sustainable open source development (Robles, Amor, Gonzalez-Barahona, & Herraiz, 2005; Negoita et al., 2019).

Not only does OSS require financial resources and software management as in business ecosystems, but there is the added component of having the project sustained by the community (Arantes & Freire, 2011). Failure in any of these areas can leave a project abandoned and thus sustainability is a base component of OSS development. In summary, any efforts that ensure ongoing day-to-day software development and its related outputs can be viewed as sustainable (Negoita et al., 2019; Ghapanchi, 2015).

Sustainability Metrics

The metrics pertaining to sustainability are organised in Table 3.2. Language variability that is present in analysing the literature show the terms productivity and engagement indicating the same construct. Engagement (as shown in Figure 3.6) is used to mean any activity that is done to sustain the software project in day-to-day operations. The second sub-classification of sustainability is general interest or popularity. Interest has no parallel in the natural ecosystem literature, but emerges as important to health and success within the software domain in many studies (Ghapanchi, 2015; Jansen, 2014; Saini, Verma, Singh, & Chahal, 2020; Wahyudin et al., 2007). To differentiate a metric between engagement and interest, ask the question: Is it an activity associated with building the project? Active participation such as commenting and submitting a pull request are considered engagement metrics. Conversely, a star or a ranking is an external statistic associated with the project.

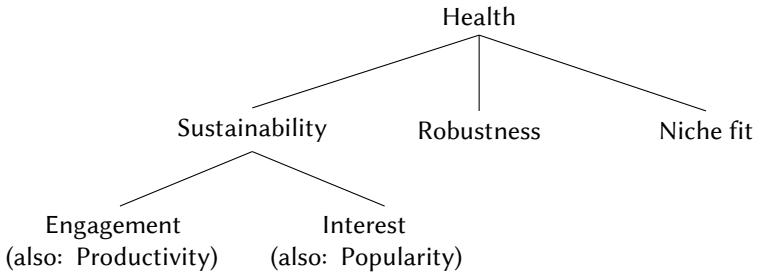


FIGURE 3.6: Summarising the language used to define health as composed of sustainability, robustness, and niche fit. Further, sustainability is composed of engagement or productivity and interest or popularity.

Tables 3.2, 3.3, and 3.4 show the metrics from the literature review, but not all of these are collected and used in the present study. Many of the metrics are lacking a suitable definition, have no hypothesised operationalisation using version control information, or are not applicable to the present study of OSS health through publicly available data. Additionally, some metrics concentrate on motivations of individuals that are very difficult to determine without targeted survey data, or are components of a business process such as marketing and financial data that does not apply or is proprietary when limiting the scope to OSS. The selection of specific metrics is a primary goal of this study and discussed further in Chapters 6 and 7.

3.3.1 General Interest

General interest or popularity involves various metrics that offer a comprehensive understanding of a project's appeal, potential for engagement, and significance within the developer and user communities. A project's popularity is often an essential factor in determining

TABLE 3.2: Metrics relating to sustainability in the OSS health literature can be split into Engagement and Interest.

Engagement		
Metric	Description	Literature[†]
Bug fix rate	How quick bugs are noted and fixed	[2, 4, 6, 7, 9, 14]
Comments	Volume; including code review, pull request, & issues-based	[4–6, 10, 13, 14]
Growth of project	Size of project increasing	[4, 7]
Growth of community	Size of community increasing	[4]
Issues	Count of issues	[6, 8, 10, 14]
Issues opened	Count of tickets	[1, 11]
LOC[*]	Size of codebase	[8, 12]
LOC added/removed	Change in size of codebase	[6, 8]
Method count	Distinct methods coded	[8]
Token count	Individual units of code	[8]
Commits	Count of commits to codebase	[3, 4, 8]
Developers	Count of authors of code	[1, 2, 4, 15]
Contributor	Count of contributors, includes developers	[4, 6, 8, 11, 13]
Developer time spent	Productivity measure of hours	[4]
Developer churn	New developers entering the project/SECO	[3, 12]
Pull requests	Count of PRs initialised	[1, 10, 13]
Releases	Count of version releases published	[3, 9]
Financial resources	Related to business operations	[4, 6]
Interest		
Dependencies	The number of software dependencies a project has, for example, Bitcoin relies on the GCC compiler collection	[11]
Downloads	Count of total downloads of a software package	[3, 6, 11, 14]
Forks	Count of number of times the software is forked	[6–8, 11]
Online rank	Ranking of the project in the broader web, for example, number of search engine hits, or Alexa page ranking	[2, 4, 6, 11]
Stars[‡]	Count of total stars on GitHub	[7, 11, 13]
Tags	Total number of tags given to a project	[11]
Watchers[‡]	Count of total watchers on GitHub	[8, 11, 13]

[*] LOC is lines of code.

[‡] Stars and watchers are exclusive GitHub event types.

[†] Source literature: [1] Chengalur-Smith et al. (2010), [2] Crowston, Howison, and Annabi (2006), [3] Ghapanchi (2015), [4] Goggins et al. (2021), [5] Hata, Novielli, Baltes, Kula, and Treude (2022), [6] Jansen (2014), [7] Negoita et al. (2019), [8] Osman and Baysal (2021), [9] Raja and Tretter (2012), [10] Robinson, Deng, and Qi (2016), [11] Saini et al. (2020), [12] Shaikh and Levina (2019), [13] Tamburri, Palomba, Serebrenik, and Zaidman (2019), [14] Wahyudin et al. (2007), [15] Z. Wang and Perry (2016).

its sustainability, as it reflects the degree of support and investment from the broader community whether time invested by volunteers or financial investment. These metrics not only provide an insight into the project's current status but also help predict its future trajectory and potential for growth. By examining metrics such as dependencies, downloads, forks, online rank, stars, tags, and watchers, one can better understand the factors contributing to a project's popularity and identify areas for improvement.

Dependencies: The number of software dependencies a project has, such as Bitcoin's reliance on the GCC compiler collection (Saini et al., 2020).

Downloads: The total count of downloads for a software package (Ghapanchi, 2015; Jansen, 2014; Saini et al., 2020; Wahyudin et al., 2007).

Forks: The number of times the software is forked, indicating active development and adaptation (Jansen, 2014; Negoita et al., 2019; Osman & Baysal, 2021; Saini et al., 2020).

Online rank: The project's ranking in the broader web, determined by factors such as search engine hits or Alexa page ranking (Crowston et al., 2006; Goggins et al., 2021; Jansen, 2014; Saini et al., 2020).

Stars: The total count of stars on GitHub, signifying user appreciation or endorsement of a project (Negoita et al., 2019; Saini et al., 2020; Tamburri et al., 2019).

Tags: The total number of tags assigned to a project, indicating classification and organisation (Saini et al., 2020).

Watchers: The count of total watchers on GitHub, reflecting the community's ongoing interest in the project's development (Osman & Baysal, 2021; Saini et al., 2020; Tamburri et al., 2019).

3.3.2 Engagement

The concept of engagement in open source software projects is a critical aspect of their overall health. In the context of OSS, engagement can be observed at various levels, including the broader community, individual projects, and individual developers. Researchers often focus on one of these levels to study engagement, with some concentrating on community-level engagement, others on individual developers, and the present study, on project-level engagement. Project-level engagement is inclusive of individual-level engagement but does not necessarily encompass an entire community's engagement or capture the same characteristics of a community.

Developer engagement as a stand alone construct is known to be important (Poba-Nzaou & Uwizeyemungu, 2019; Fang & Neufeld, 2008; Shaikh & Levina, 2019; Tamburri et al., 2019) as it reflects the ongoing contributions and collaborative efforts of developers and other community members towards a common goal but does not itself have a clear index of the factors that make up engagement. Software community engagement is a subjective term referring to how people interact and contribute to a project, and includes activities both coding and non-coding related: managing the community, development, documentation, and participating in discussions (Z. Wang, Feng, Wang, Jones, & Redmiles, 2020).

Community Level Engagement

Community engagement is a key characteristic of OSS (Tamburri et al., 2019) and refers to all the possible roles members may play within the ecosystem and how they contribute. This contribution is through their activity which, at the top level, is a comprehensive umbrella for any project-related effort. There is considerable role diversity to community engagement, from that of core developer and member of the organisation, to that of a casual user that downloads a package, to the curious individual that bookmarks an interesting project by giving it a star on GitHub. Additionally, watchers, forkers, committers, commenters, issue reporters, fork committers, pull-requesters, members, and coordinators all add to engagement (Z. Wang et al., 2020). Community engagement is thus the aggregate of these members' contributions, which can be applied at the project level or higher at the ecosystem level. Community engagement is summarised by a count of artefacts such as bugs fixed or responses to forum posts (Daniel, Agarwal, & Stewart, 2013), although this excludes many possible avenues and lacks a rigorous metric selection process.

Although the present concentration is on the project level within the

$$\text{ecosystem} \rightarrow \text{project} \rightarrow \text{developer}$$

hierarchy, much of the research on individual developers is relevant as it is necessary to collect metrics composed on single units that can be aggregated to a whole project.

Individual Level Engagement

Individual developers contribute in a variety of ways, with diverse motivations driving their participation, often without financial incentive (Bosu, Iqbal, Shahriyar, & Chakroborty, 2019). Community groups may or may not have a structured organisation or well-defined roles. Relying solely on developer activity as an indicator of software health can be misleading, as it only represents a piece of the health puzzle (Link & Germonprez, 2018). The ease of obtaining code

metrics can contribute to this issue. In developing an ecosystem health framework, Jansen (2014) acknowledges that ecosystem health does not necessarily equate to project health and that individual project health is often determined by calculating various activity metrics.

Researchers use different terms to represent developer interest, such as *developer attraction*, *team size*, *developer interest*, *development base*, and *developer join* (Ghapanchi, 2015). Interest can lead to participation and engagement, which is referred to as sustained developer participation (Fang & Neufeld, 2008), developer commitment (Schilling, 2014), and activity (Link & Germonprez, 2018).

Measuring individual developer engagement is approached through subjective and survey-intense methods like counting the hours worked per week (Poba-Nzaou & Uwizeyemungu, 2019). However, a developer's time is not spent solely on a single activity; OSS contributors engage in various tasks, including coding, managing the community, working on parallel projects, creating documentation, and participating in discussions (Z. Wang et al., 2020).

Although individual motivation for contributing to OSS is an interesting area of study, software is rarely produced by single coders, and given the ease of access to forking and pull-request based improvement it is more representative to use a project level context.

By focusing on project-level engagement, there is a better understanding of the dynamics and contributions of multiple developers within a specific project, providing valuable insights into the factors that contribute to the health of OSS projects. In the context of blockchain projects, engagement is particularly critical given the decentralised nature of the technology and the reliance on collaborative efforts to maintain and improve the software. This understanding can, in turn, inform best practices and strategies for fostering engagement and improving the health of OSS projects within the blockchain ecosystem and beyond.

3.4 Robustness

Once sustainability is established the ecosystem can remain productive over time only if it can sustain shocks that threaten its viability. Biodiversity is important to natural ecosystems and helps to enable recovery from shocks. This is through many species performing the same function such as photosynthesis or decomposition, and also by individual species having unique environmental response to threats when compared to their close relatives (Dhungana et al., 2010). Costanza (1992) and D. Rapport et al. (1998) both identify resilience as the ability of an ecosystem to overcome disruption to its local environment.

Environmental factors affect all members of the ecosystem and are generally out of control of the community itself. External risks are more difficult to identify such as misaligned product-market fit, the competitor landscape, technological innovation, and the regulatory and legal landscape (Chengalur-Smith et al., 2010). Many of these dramatic shocks are difficult to quantify and out of scope when considering project level software.

Robustness Metrics

In the context of software ecosystems, robustness is a key attribute that contributes to the resilience, stability, and overall health of projects within the ecosystem. As seen in Table 3.3, various metrics from the literature contribute to robustness within a SECO. These metrics encompass a wide range of aspects, including project age, business metrics, code quality, contributor and end user characteristics, developer centrality, and more.

Age of project: The duration since the project repository's creation, which can provide insights into its stability and maturity (Chengalur-Smith et al., 2010; Goggins et al., 2021; Osman & Baysal, 2021).

Business metrics: These encompass management, process development, and systems development aspects of a project, which can influence its overall robustness (Goggins et al., 2021).

Code quality: Relates to code metrics, such as cyclomatic complexity, which can impact a project's maintainability, efficiency, and reliability (Goggins et al., 2021; Osman & Baysal, 2021).

Contributor characteristics: Includes factors like centrality, reputation, and satisfaction of contributors, which can influence the project's long-term success and resilience (Jansen, 2014).

Developer centrality: Measures the connectedness of developers to other developers across different projects, affecting knowledge sharing and collaboration (Wahyudin et al., 2007).

Developer longevity: The amount of time a developer contributes to a single project, providing insights into their commitment and the project's ability to retain talent (Goggins et al., 2021).

Developer organisation count: The number of organisations a developer engages with, potentially reflecting their expertise and network within the ecosystem (Jansen, 2014).

Geographic distribution: The extent of global contribution distribution, indicating the project's accessibility and appeal to developers from various regions (Tamburri et al., 2019).

Interoperability: The costs associated with switching between Software Ecosystems (SECO), which can impact the project's adaptability and ease of integration (Jansen, 2014).

Knowledge creation: The degree of knowledge addition to the SECO and artefact creation, contributing to the project's innovation and growth (Jansen, 2014).

Market share: The ratio of users to the total local ecosystem, reflecting the project's popularity and influence within the specific ecosystem (Jansen, 2014).

Network centrality: The connections to the wider SECO, demonstrating the project's integration with other projects and stakeholders (Jansen, 2014).

Partnership centrality: The number of partners and connectedness with partners, indicating the project's collaborative potential and support network (Jansen, 2014).

Project centrality: The connectedness of a project to other projects in the ecosystem, highlighting its integration and potential for collaboration within the ecosystem (Jansen, 2014; Tamburri et al., 2019).

Truck factor: The project risk due to dependency on single or few contributors, potentially affecting the project's resilience and long-term stability (Goggins et al., 2021).

End user metrics: These include factors such as user count, longevity, loyalty, and satisfaction, which can provide insights into the project's reception, adoption, and ongoing support by its user base (Goggins et al., 2021; Jansen, 2014; Z. Wang & Perry, 2016).

In summary, robustness in software encompasses various demographic factors, such as the age and size of the population or the software organisation. Both of these factors are positive indicators—long-time contributors are more likely to continue contributing, and long-standing projects have demonstrated their ability to survive and adapt to potential shocks. Resilience can manifest through the geographic distribution of contributors, mitigating some of the environmental risks affecting all participants, and through market share information that provides external validation for the project. Internal validation within the community arises from the utilisation, incorporation, and reliance on the software by others, which can be quantified using a criticality measure (Arya, Brown, & Pike, 2022).

It is important to acknowledge that numerous robustness measures, such as code quality and user satisfaction, may prove challenging to assess or are inherently subjective in nature. The determination of these metrics is further explored in Chapter 7.

Table 3.3: Metrics relating to robustness in the OSS health literature. Many robustness measures are difficult to gauge or subjective in nature such as code quality and user satisfaction.

Robustness		
Metric	Description	Literature[†]
Age of project	Time since repository creation	[1, 2, 3]
Business metrics	Including management, process development, and systems development	[2]
Code quality	As relates to code metrics such as cyclomatic complexity	[2, 3]
Contributor characteristics	Including centrality, reputation, and satisfaction	[4]
Developer centrality	Connectedness to other developers in other projects	[5]
Developer longevity	Time spent contributing to a single project	[2]
Developer org count	Number of organisations a developer engages with	[4]
Geographic distribution	Global contribution distribution	[6]
Interoperability	Costs of switching SECO	[4]
Issues fix rate	Amount of time taken to fix bugs and close issues	[8,9]
Knowledge creation	Knowledge added to SECO and artefact creation	[4]
Market share	Ratio of users to the total local ecosystem	[4]
Network centrality	Links to the wider SECO	[4]
Partnership centrality	Number of partners and connectedness with partners	[4]
Project centrality	Connectedness to other projects in the ecosystem	[4, 6]
Truck factor	Project risk due to single or few contributors	[2]
End user metrics	Including count, longevity, loyalty, and satisfaction	[2, 4, 7]

[†] Source literature: [1] Chengalur-Smith et al. (2010), [2] Goggins et al. (2021), [3] Osman and Baysal (2021), [4] Jansen (2014), [5] Wahyudin et al. (2007), [6] Tamburri et al. (2019), [7] Z. Wang and Perry (2016), [8] Negoita et al. (2019), [9] Raja and Tretter (2012).

3.5 Niche Fit

A third characteristic in the definition of health (Figure 3.6) is that of occupying a relevant niche in the ecosystem. Diversity of ecology is important to natural ecosystems to enable species to recovery from dramatic events. This is through many species performing the same function such as photosynthesis or decomposition, and also by individual species having unique environmental response to threats when compared to their close relatives (Dhungana et al., 2010). Niche occupancy means at least a single species exists at all levels and across all functions of the aggregate system.

This is paralleled as niche creation in business ecosystems (Iansiti & Levien, 2004), and software ecosystems (Jansen, 2014). Ongoing competition requires projects in areas with many similar products to pivot and search for a niche to be successful. At the software project level product fit can be quantified by audience niche, programming language niche, and operating system niche (Chengalur-Smith et al., 2010). To have the best chance at occupying a niche, a project may also choose to support multiple natural languages, push applicability to a variety

of markets, and be open to various contributor roles (Jansen, 2014). These metrics are shown in Table 3.4.

Niche Fit Metrics

Niche fit plays a role in determining how well a project thrives within its ecosystem. The specific metrics that aid in measuring niche fit are further detailed in Table 3.4.

SECO project variety: The diversity of projects within an ecosystem, showcasing the range of available niches. A higher variety of project types indicates a more diverse ecosystem, offering numerous opportunities for collaboration and innovation (Jansen, 2014).

Programming language: By supporting a variety of programming languages, a project can accommodate new contributors with diverse programming skill sets. This flexibility broadens the project's appeal and facilitates the inclusion of a wider range of developers (Chengalur-Smith et al., 2010).

Natural language: The diversity of natural languages within a project enables the involvement of contributors from various linguistic backgrounds. This inclusivity fosters a more globalized and accessible project environment (Jansen, 2014).

Market variety: A project's applicability across different markets is indicative of its versatility and potential for expansion. By targeting multiple markets, a project can demonstrate adaptability and increased utility, thereby appealing to a broader range of stakeholders (Jansen, 2014).

Operating system: Compatibility with a variety of operating systems allows projects to attract new contributors who utilize different computing environments. This metric emphasizes the importance of accessibility and inclusivity in project development (Jansen, 2014; Chengalur-Smith et al., 2010).

Contributor types: The availability of diverse contributor roles within a project ensures that individuals with varying skills and expertise can participate. This variation encourages collaboration and fosters a dynamic project ecosystem (Jansen, 2014).

AMI: Average Mutual Information. This metric quantifies task specialization and the coordination of specialists within a project. A higher AMI value suggests that the project effectively leverages the skills and expertise of its specialists, thereby increasing the overall efficiency and output (Raja & Tretter, 2012).

Table 3.4: Metrics relating to the niche occupancy of a project within an ecosystem in the health literature.

Niche fit		
Metric	Description	Literature[†]
SECO project variety	Variety in types of projects in the ecosystem demonstrating available niches	[1]
Programming language	Support for a variety of languages allows for new contributors to participate	[2]
Natural language	Variety of natural languages allows for new contributors	[1]
Market variety	Applicability of the project to different markets	[1]
Operating system	Variety of OS allows for new contributors	[1, 2]
Contributor types	Variation in available contributor roles	[1]
AMI[*]	A measure of task specialisation and the coordination of specialists	[3]
Niche size	More member organisations within a niche add legitimacy	[2, 4]

[*] AMI is average mutual information
[†] Source literature: [1] Jansen (2014), [2] Chengalur-Smith et al. (2010), [3] Raja and Tretter (2012), [4] Goggins et al. (2021).

Niche size: The number of member organizations within a niche contributes to its legitimacy. A larger niche size can enhance credibility, attract additional resources, and facilitate collaboration among stakeholders (Chengalur-Smith et al., 2010; Goggins et al., 2021).

It is essential to note that niche fit metrics necessitate a comprehensive view of the entire ecosystem. By concentrating on individual projects it is difficult to contextualise a project in the wider landscape and thus niche fit is an aggregate measure that requires a broad view of the whole ecosystem. Therefore, niche related metrics are not within the scope of the study.

3.6 Blockchain Health

Returning to the history of open source software shown in Figure 3.1, Bitcoin is born in the OSS 2.0 era characterised by corporations embracing open source. Since its first release in 2009 Bitcoin has inspired a burgeoning industry in blockchain software development with decentralisation as a core value, starting the wave of OSS 3.0. Numerous projects associated with the blockchain industry have emerged, as extensively documented by CoinMarketCap since 2012. At present, CoinMarketCap tracks over 20,000 tokens and projects, all connected to blockchain technology. Blockchain software, as defined by the collection of various token projects from CoinMarketCap—including cryptocurrencies, platforms, protocols, decentralised web3.0 applications, stablecoins, and support libraries of smart contracts—differs significantly from other software industries in the landscape (Bosu et al., 2019).

The open source nature of these projects enables developers to select those that align with their ideological preferences (Smirnova, Reitzig, & Alexy, 2022). More developers cite motives for contributing based on a "bitcoin ideology" than their non-blockchain counterparts (Bosu et al., 2019; Hars & Ou, 2001). This ideology emphasises a highly decentralised environment, aligning with permissive, open licences that encourage forking.

The token-based structure of blockchain projects further facilitates incentive-based participation. While OSS relies on voluntary contributions, most developers receive compensation through grants, scholarships, or indirect benefits from the token economy. By improving a project, developers can directly profit from token appreciation, which is not feasible in other OSS domains such as Android or Linux.

Compared to non-blockchain projects, blockchain software places greater emphasis on security and reliability (Bosu et al., 2019). Moreover, it involves more complexity and requires distinct tools. Consequently, the cost of defects is higher in this high-risk, high-reward field, which tends to attract younger developers (2019).

Research on health in blockchain projects and ecosystems is limited. Osman and Baysal (2021) categorise health metrics in the Bitcoin ecosystem as: popularity, complexity, activity, and age. However, the study's primary indicators are heuristic-based and lack solid rationale. No known studies investigate the health of other blockchain projects, such as Ethereum or Solana, or broader collections of projects and ecosystems. This research gap concerning the health of blockchain software at both the individual developer and software project levels provides an opportunity for exploration in the present study, guided by the following research questions.

3.7 Research Questions

Chapter 2 reviews the literature in blockchain systems identifying the relationship between decentralisation and consensus in the blockchain trilemma (Section 2.4) which results in the following research question posed in Section 2.5: What are the factors that influence blockchain consensus?

The goal of reviewing software health in an open source context is to lay the groundwork for developing a model of software health in the blockchain field and address the literature gap that supports the following research questions:

Research Question 2

What is a definition of software health?

RQ2a

What metrics express open source software health factors?

RQ2b

What is the nature of the relationship between factors influencing software health?

The third research question ties the study of blockchain software together with software health.

Research Question 3

What is included in a comprehensive model of blockchain software health?

These questions guide the investigation to fill a gap in the literature at the intersection of blockchain and software health.

3.8 Conclusion

Open source software has changed they way software applications are created, updated, and published. The collaborative nature in combination with ubiquitous version control software has led to the availability of trace data for every aspect of the software creation process. However, not all collaborative efforts yield the same results, necessitating the establishment of metrics and the identification of trends that indicate project health. Blockchain software, a product of the open source movement, marries the incentives of the token economy and blockchain values with permissive licensing, creating a unique landscape for investigation.

Drawing on parallels from natural ecosystems, three components of health are proposed: sustainability, robustness, and niche fit. Sustainability in the software context refers to the ongoing efforts of contributors that propel a project towards its objectives. Robustness characterises a project or team's resilience in the face of unexpected challenges. Niche fit describes the degree to which a project successfully fulfils a market need by developing useful software. Despite the burgeoning blockchain ecosystem, there is a lack of health studies in this area, leaving key research questions unanswered: How can factors be identified that affect block-

chain health? What does a model of blockchain software health look like? And, how does the consensus mechanism impact the health of blockchain projects?

Next, in Chapter 4, the research design is detailed including the methodology to address the research questions.

Chapter 4

Research Design

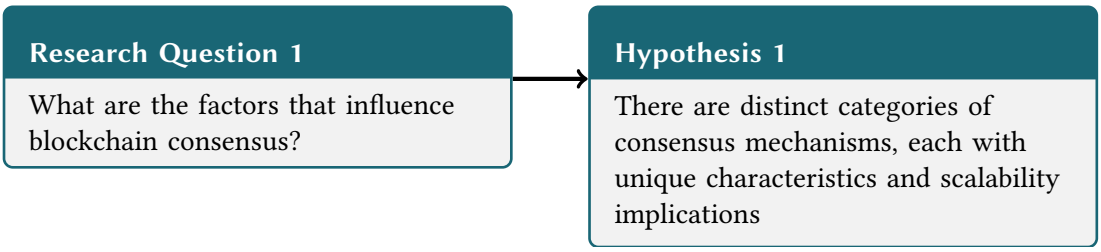
> *Scientific theories are universal statements. Like all linguistic representations they are systems of signs or symbols. Theories are nets cast to catch what we call 'the world'; to rationalize, to explain and to master it. We endeavor to make the mesh even finer and finer.*
>
> Karl Popper
> *The Logic of Scientific Discovery*
> 1934

Chapters 2 and 3 Review the relevant literature to highlight the research gap where the problem rests. The broad aim of the research is to (i) define and identify factors that contribute to open source blockchain project health and (ii) to develop a framework to enable practitioners to identify and maximise the health of projects.

To get there, research questions formulated as the basis for the thesis are next in Section 4.1. The chapter discusses the rationale leading to Design Science as a research methodology (Section 4.2), followed by the specific testing methods in Section 4.3, and conclusions in Section 4.4.

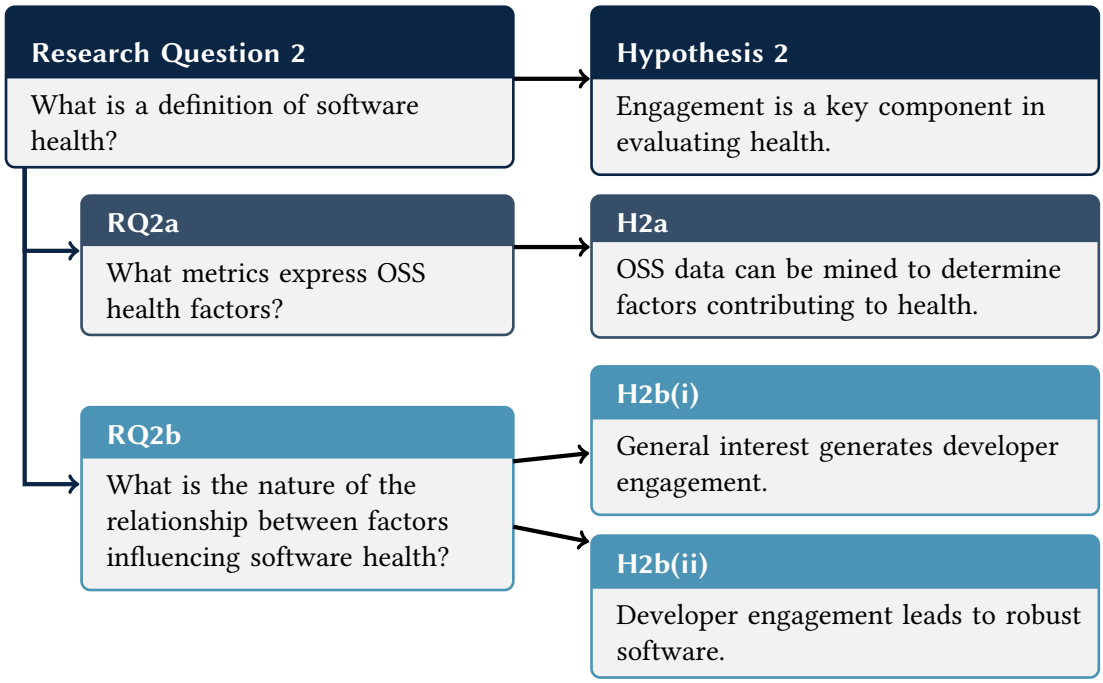
4.1 Research Questions

The tension within the blockchain trilemma described in Section 2.4 is between blockchain scaling and decentralisation, the latter which holds a strong link to the underlying consensus method. The consensus method is tailored by design with performance as an outcome that is linked to scalability. If performance is considered good then scaling would not be necessary, however the research into consensus methods suggests otherwise. RQ1 aims to investigate the core consensus methods and establish the inherent design factors.

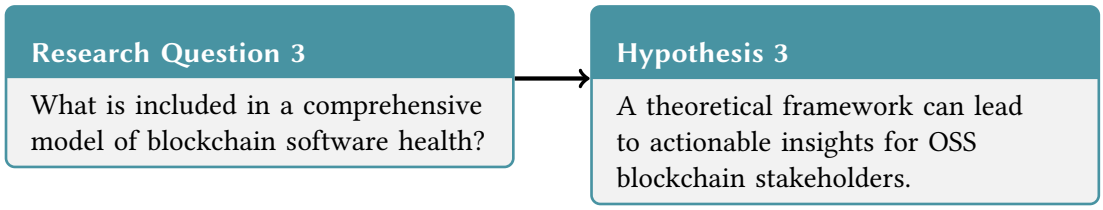


RQ1 is investigated by testing the hypothesis: There are distinct categories of consensus mechanisms, each with unique characteristics and scalability implications.

Chapter 3 identifies the lack of a cohesive definition of software health and the often missing practical methods of selecting, gathering, and analysing data. Research question two is concerned with software health and spawns two additional sub-questions.



Motivated by models of ecosystem health combined with prior models of software health (Section 3.2.3), the third research question ties together the research and generates a framework for blockchain software health.



RQ3 is investigated by testing the hypothesis: H3—A framework of blockchain software health can inform and guide researchers and practitioners towards improved OSS. In other words, highlighting areas that contribute to blockchain health can be a key step in improving software.

Figure 4.3 summarises the research questions including their associated hypotheses that are investigated in this work and the individual methodologies.

4.2 Research Design

The research design, working from the general to the specific, in an approach that lends itself to the layers of the onion analogy, is discussed here (Saunders, Lewis, & Thornhill, 2007). Beginning on the outside in Figure 4.1 the broad philosophical alignment is discussed.

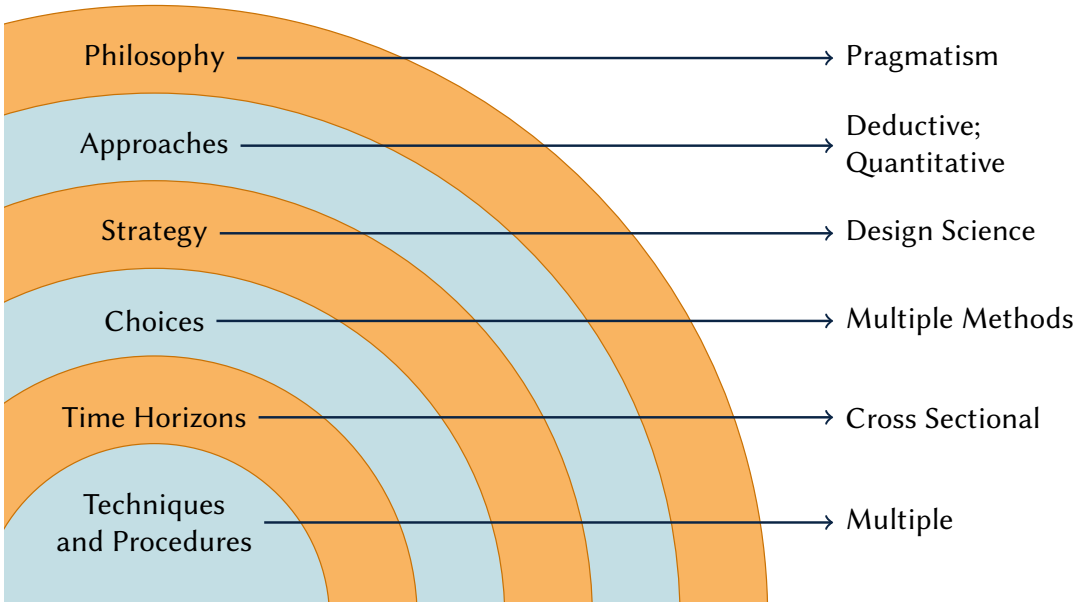


FIGURE 4.1: The overall research design organisation beginning with the outer layer–research philosophy, and progressing inwards towards techniques. Adapted from Saunders et al. (2007).

4.2.1 Research Philosophy - Pragmatist

Situated at the outermost layer of the research onion is the research philosophy, which encapsulates the belief system guiding the collection, analysis, and interpretation of data. This layer includes perspectives such as positivism, interpretivism, and realism.

Positivism, anchored in an ontological perspective asserting a singular reality, originates from the natural sciences. It posits that observable phenomena provide the only authentic source of meaning, with the positivist researcher embodying the role of a detached observer arriving at a single, unambiguous conclusion (Walliman, 2011). For instance, the hypothesis, *'computer memory is finite'*, can be independently confirmed, exemplifying positivism's empirically verifiable stance.

In contrast, interpretivism acknowledges the complexity of multiple, co-existing worldviews, where different observers might draw different conclusions. It advocates an empathetic approach, acknowledging that knowledge is created through a shared understanding or consensus, whether partial or complete (Flick, 2014).

This research leans towards an interpretivist philosophy. While socio-technical research in IS possesses several positivist elements, the incorporation of a social component necessitates an interpretivist stance. However, given the research's objective of producing a tangible artefact to augment academic knowledge and assist practitioners, a pragmatist per-

spective synthesising these philosophical views is adopted. As a research philosophy, pragmatism emphasises action, change, and solving practical problems as its primary driving forces (Goldkuhl, 2012). It is a flexible approach that offers researchers a rich framework for investigations, facilitating a dynamic and adaptable perspective that is well-suited to real-world applications.

4.2.2 Research Approach – Deductive

At the next layer of the research onion, the research approach can broadly be categorised into two types: inductive and deductive. The choice between these two paths depends on the existing body of knowledge and the nature of the research question being addressed.

In an inductive research approach, data collection comes first, serving as the foundation for theory development. This approach is typically applied when there is limited pre-existing theory relating to the research question or when a fresh perspective is sought. It's an exploratory path that begins with specific observations and measures, and ends with broader generalisations and theories (Walliman, 2011).

On the other hand, a deductive approach starts with a theory, forms a hypothesis, and then works towards its confirmation or rejection through systematic data collection and analysis. This approach is typically associated with scientific research, as it offers a structured methodology that is particularly suitable when the existing theory is robust (Walliman, 2011).

The research approach for the present study aligns with the deductive path. This is due to the established theoretical foundation within the field of IS upon which this work is built (Gregor, 2006; Beck, Weber, & Gregory, 2013). Starting from this theoretical basis, the research seeks to test specific hypotheses and contribute further to this existing body of knowledge.

4.2.3 Design Science in Information Systems

The next layer of the research design onion is the research strategy, which outlines the pathway to be followed to attain the research objectives. This study employs a Design Science Research Methodology (DSRM), a well-recognised research strategy in the IS field (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007; Offermann, Levina, Schönherr, & Bub, 2009). Alternate methods are discussed in Section 4.2.7.

Design science centres on crafting artefacts to address specific needs or achieve set objectives. These artefacts may take various forms, such as software, algorithms, processes, or methods. The creation of the artefact comprises the design activity, while the introduction of novel knowledge to the field embodies the scientific research component. This distinguishes it

from traditional design, which applies existing knowledge to organisational problems through artefacts (Kotzé, Van Der Merwe, & Gerber, 2015). The research aspect introduces procedural rigour with the explicit goal of expanding the knowledge horizon. Whereas standard design may seem arbitrary to the observer, or even the participant, design science is underpinned by a logical approach.

Design science made significant strides in 2004 when it is applied to IS research, gaining momentum with the publication of *Design Science in IS Research* by Hevner, March, Park, and Ram (2004). The synthesis of design and science with a research component provides a systematic and measured approach to bridge knowledge gaps (Vaishnavi, Kuechler, & Petter, 2004).

IS research often applies theories from related fields like computer science and economics to technology and organisational issues (Peffers et al., 2007). A design-based methodology situates itself at the nexus of design research and applied science, making IS a fitting domain (Offermann et al., 2009). Blockchain research, straddling computer science, mathematics, and economics, finds applications in various industries, from banking to gaming, offering social benefits like privacy and transparency. Hence, a design science methodology is apt for a blockchain research problem. The DSRM proposed by Peffers et al. (2007) amalgamates recent developments in design science with IS research. It employs a systematic framework to a known problem, aimed at enhancing output and refining experiences through iterative processes. A research methodology framework needs to delineate principles, prescribe guidelines for research conduct, and propose a robust procedure for output generation. This method mirrors those in engineering and computer science, where a need is identified and a solution derived through the design process. The model comprises six stages and incorporates elements from Hevner's seven guidelines for design science research (Hevner et al., 2004).

The choice of the DSRM proposed by Peffers et al. is motivated by its ability to accommodate both problem- and objective-centred research entry points, a feature that suits the multifaceted nature of this study. As depicted in Figure 4.2, this model allows for multiple entry points and the possibility for numerous iterative cycles from different phases. The first entry point of this research adopts a problem-centric view: blockchains have difficulty scaling. The remaining entry points are objective-centric, focusing on defining OSS health and developing a model of blockchain project health.

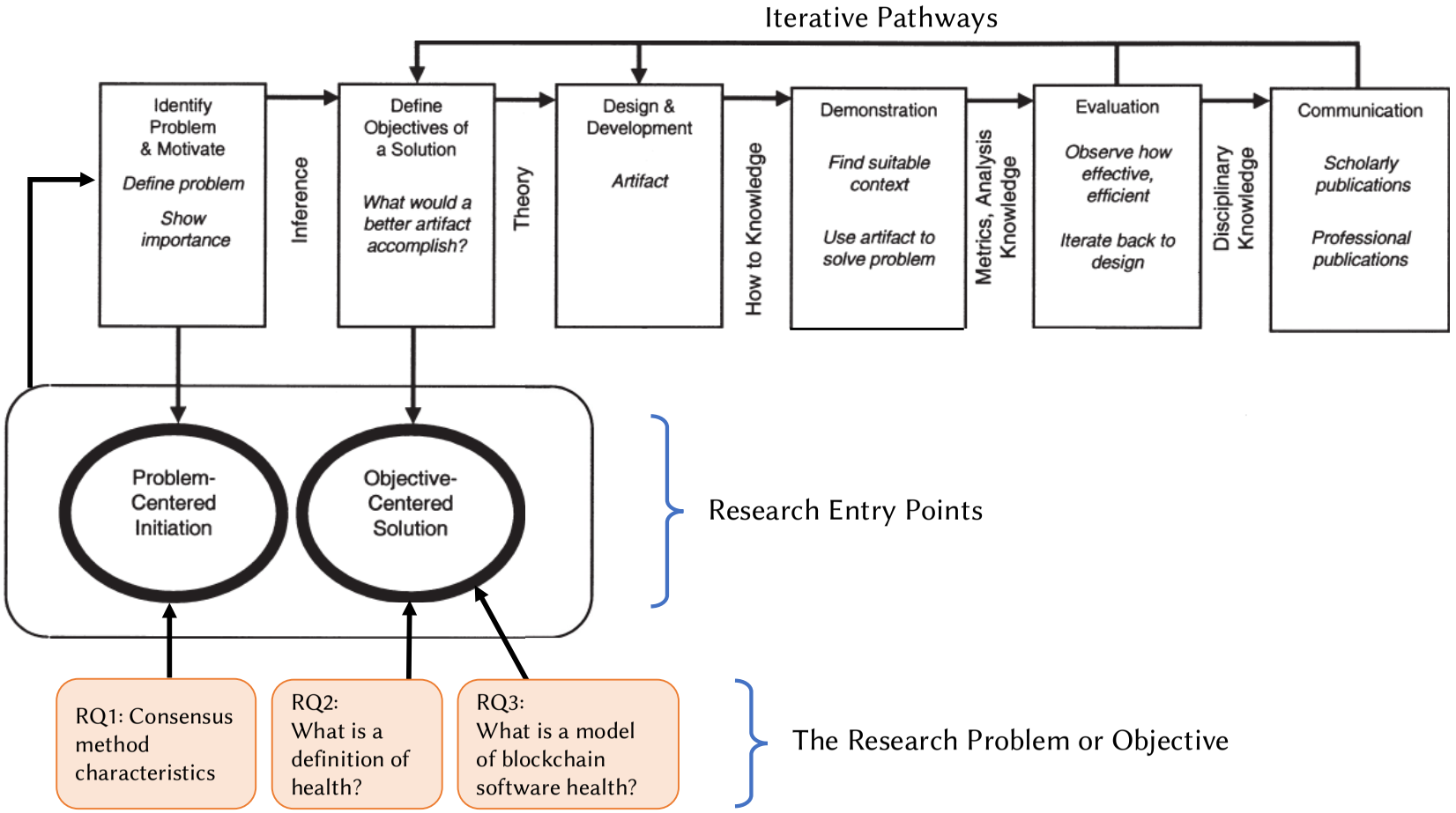


FIGURE 4.2: Design Science Research Methodology. The problem and objective statements are related to the research questions and serve as entry points for the research. Iteration is expected from the evaluation and communication stages. Modified from Peffers et al. (2007).

4.2.4 Research Choices – Multiple Methods

Advancing to the next layer of the research onion, the choice between mono-method, mixed-method, and multi-method research design is made. This layer concerns the researcher's decision on the type of data to be collected and the methods used for analysis: quantitative, qualitative, or a combination of both.

For this study, a multi-method research design is selected, using quantitative data. This approach provides a comprehensive and nuanced understanding of the research problem in alignment with DSRM principles. The specifics of this multi-method design are described in Section 4.3.

4.2.5 Research Time Horizon – Cross Sectional

Moving inwards, the next layer of the research onion considers the time horizon of the research. This can be either cross-sectional, where a particular phenomenon is studied at a specific point in time, or longitudinal, where the study spans across a certain period.

For this study, a cross-sectional time horizon is adopted, akin to the approach used in the statistical studies (Chapters 6 to 8). The focus is to discern the model structure for software health as of March 2022. Historical data is gathered from this point, reaching back to the inception of each project, in some instances covering more than five years.

While a longitudinal perspective could yield valuable insights into the evolution of projects over time, practical constraints such as time limit the feasibility of this approach in the current research context. Therefore, the cross-sectional time horizon provides a feasible and effective framework for investigating the research questions at hand.

4.2.6 Research Techniques and Procedures

The core of the research onion pertains to the specific procedures utilised for data collection and data analysis. This study employs multiple techniques to address the research questions. It commences with the development of a taxonomy, as detailed in section Section 4.3.1. This is followed by an exploratory factor analysis (Section 4.3.2), structural equation modelling (Section 4.3.3), and culminates in the development of a comprehensive framework (Section 4.3.4).

4.2.7 Alternative Methods – Evaluation & Exclusion Rationale

While various research methodologies within IS are considered, they are ultimately deemed unsuitable for this study. The following section provides an overview of these methodologies,

further underscoring the rationale behind the selection of DSRM.

Positivist Case Studies

Positivist Case Studies are a research method that adopts the principles of positivism to explore, understand, and explain phenomena within specific contexts. Positivism, as a philosophical stance, assumes that reality is objective and can be discovered through empirical observation and logical analysis. It can employ both quantitative or qualitative methods, formulating clear hypotheses, and seeking causal relationships or patterns that can be generalised (Dubé & Paré, 2003). A positivist case study might involve studying the implementation of an information system in an organisation, with the aim of identifying factors that contribute to its success or failure. The study would likely involve collecting objective data (for example, system usage statistics, project timelines, cost figures), testing hypotheses (for example, "User training increases system adoption"), and seeking to generalise findings to other similar implementations.

However, the underlying principles of positivist case studies can be at odds with DSRM. While positivist case studies typically serve confirmatory research, testing existing hypotheses or theories, DSRM employs exploratory research to design and build novel artefacts, subsequently evaluating their utility or effectiveness. Moreover, the development of artefacts in DSRM is an inherently researcher-intensive task, contrasting with the detached observer role often associated with positivist case studies (Shanks, 2002). Therefore, DSRM emerges as the more suitable approach in this context.

Interpretive Research

Interpretive Research is a perspective that prioritises the context of the information system and its interactions with that context (Klein & Myers, 1999). Interpretivist case studies often necessitate researchers to immerse themselves in a social environment, aiming to understand the meanings that members associate with phenomena. Considering software health, an interpretivist field study would be more fitting to comprehend a single project over an extended time duration, rather than attempting to generalise across a large population such as all open source blockchain projects. Thus, interpretive stances are not considered.

Action Design Research (ADR)

ADR proposes a critique of traditional design research, arguing that it overemphasises artefact design at the expense of considering the organisational context in which the artefact is created and used. ADR posits that design within organisations often involves the adaptation of existing artefacts, an approach that may lack the rigour associated with academic design (Sein, Henfridsson, Purao, Rossi, & Lindgren, 2011). This nuance is typically overlooked in design science due to the clear demarcation between evaluation and construction phases. ADR underscores the ensemble artefact as a core principle of IS research, wherein the

structures of the organisational domain are embedded in the artefact during its development and use (Orlikowski & Iacono, 2001).

However, this present study, which is set outside the organisational context of OSS projects, finds little applicability for ADR. The focus of this study aligns more closely with the principles of DSRM, which concentrate on the creation and evaluation of novel artefacts. The ensemble view of ADR, which integrates the artefact with its organisational context, may not be as relevant in a study like this, which does not centre on an organisational setting. As such, traditional design research, particularly DSRM, emerges as the more suitable approach for this study.

4.3 Research Methods

The testing of hypotheses necessitates the use of robust methods and techniques. This is the focal point of the layered structure in Figure 4.1 and where the generation of new knowledge takes place. It is crucial that these methods are grounded in empirically validated theories. Every method used in this research is presented along with a rationale for its selection, as well as a detailed procedure for data collection and analysis. The specifics of implementation, the results derived, and the validation of these findings are discussed in the subsequent Chapters 6 to 8. A comprehensive visualisation of the entire research design is in Figure 4.3.

4.3.1 Taxonomy Development

Blockchain performance is tied to the consensus method is the first hypothesis (Figure 4.3). This hypothesis is informed by a review of the literature as detailed in Chapter 2 consisting of secondary research, mainly through peer-reviewed journal articles, conference papers, and industry reports. The review concludes there is a stagnation in the development of blockchain technology with regards to scalable solutions. To test this hypothesis a scoping study is undertaken, followed by a content analysis. The result is the development of a taxonomy of blockchain consensus methods.

The review from Chapter 2 is supplemented with a scoping study to narrow down the search and identify studies associated with blockchain consensus and performance. Arksey and O'Malley (2005) provide a five-part framework for a scoping study. The parts are: identifying the research questions, identifying the relevant studies, selection of studies, charting of data, and lastly, collating, summarising, and reporting results.

A scoping study is chosen within the broader context of design science. As defined by Davis, Drey, and Gould (2009), scoping studies "characteristically involve the development, assimilation and synthesis of broad base of evidence derived from a diverse range of research

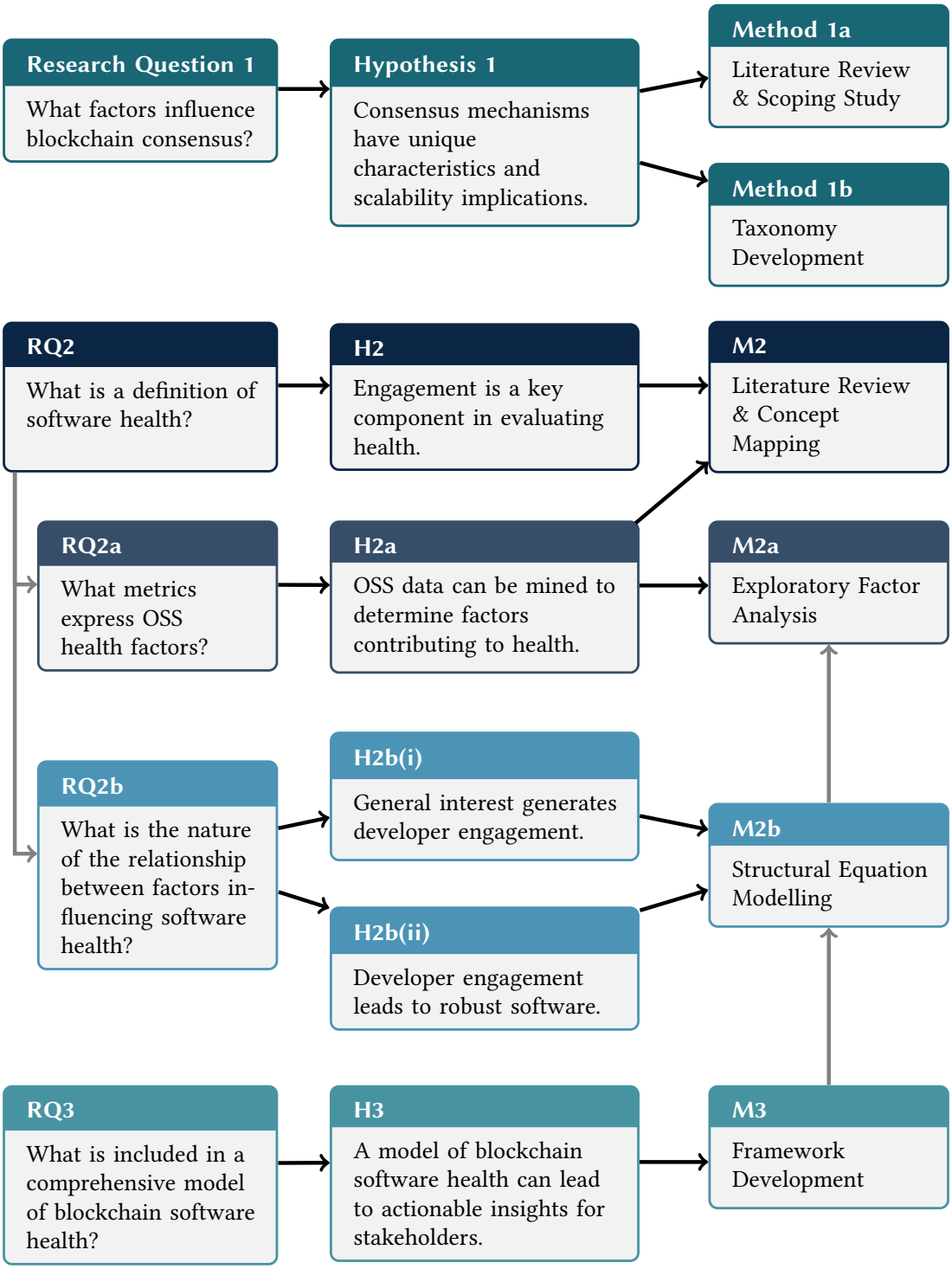


FIGURE 4.3: The research landscape. Research questions mapped to the hypotheses and testing methods. Method 3, Framework Development, relies on Method 2b, which, in turn relies on Method 2a.

and non-research sources." While scoping studies are traditionally undertaken in the health care domain, there are a number of cross over points that can be borrowed for use in computer science research (Kitchenham & Charters, 2007). The study allows for a broader range of topics and sources to be considered and does not have a quality assessment as included with a traditional systematic literature review (Okoli, 2015).

To supplement the scoping study, a content analysis is operationalised through the use of NVivo qualitative data analysis software (QSR International Pty Ltd, 2018). Content analysis, at its core, is an empirical methodology that allows for the systematic and objective interpretation of qualitative data in order to discern patterns, themes, or biases (Krippendorff, 2004). It is an advantageous tool in research areas marked by copious amounts of textual data, such as social sciences, media studies, psychology, and notably in this instance, computer science (Bauer, 2000).

One of the primary benefits of employing content analysis is its capacity to translate large amounts of data into manageable forms, allowing for concise and informed interpretation (Neuendorf, 2017). The method is flexible and adaptable to a variety of research contexts and data sources, in this case academic journal articles. Through the systematic examination of text, content analysis can yield rich information pertaining to the frequency, relationships, trends, and patterns of certain themes or concepts within the data (Elo & Kyngäs, 2008).

Taxonomy Method & Justification

To better understand blockchain consensus methods (reviewed in Section 2.3.1) a taxonomy is developed. Creating a taxonomy is a multi-stage process. First the literature is collected and assessed in a process similar to a literature review. Present taxonomies in the field of blockchain consensus are identified to ensure a meaningful contribution can be made to the field. The final stage applies the method by Nickerson, Varshney, and Muntermann (2013) for developing the taxonomy.

The taxonomy methodology proposed by Nickerson et al. (2013) demonstrates its versatility and relevance across multiple fields, thereby justifying its utilisation in the current study. The methodology is employed in the field of software engineering; for instance, Strode (2016) leveraged it to develop a taxonomy for agile software development projects. Furthermore, it demonstrates its utility in the business development sphere, specifically in the context of software-based business model development tools. Szopinski, Schoormann, John, Knackstedt, and Kundisch (2020) employ this methodology to devise a taxonomy that discerns characteristic functions of such tools, thereby promoting innovation. These applications underscore the methodology's proficiency in elucidating intricate elements and their interrelationships within complex systems.

Most importantly, for the current study, this methodology shows significant applicability within the realm of blockchain systems. Numerous scholars use the method to create taxonomies that enhance understanding of the blockchain domain. Tönnissen and Teuteberg (2020) use it to develop a taxonomy of blockchain-based applications, whilst (Sai, Buckley, Fitzgerald, & Gear, 2021) apply it to classify different levels of centralisation in blockchains. Additionally, Six, Herbaut, and Salinesi (2022) use the method to identify recurring patterns in blockchain software engineering. Collectively, these instances underscore the suitability of the method in comprehending and categorising the multifaceted aspects of blockchain systems, thus providing robust justification for its adoption in the present research.

Taxonomy Evaluation

The process of taxonomical validation, crucial in verifying the effectiveness of a new classification system, is underpinned by the application of diverse example cases (Nickerson et al., 2013). These cases, exhibiting substantial dissimilarities, should serve to unmask any potential issues with the taxonomy during its assessment. It is a critical condition, however, that these test cases must not be involved in the initial creation and refining process of the taxonomy. This evaluation's primary purpose is to ascertain the fit between the cases and the taxonomy, and to identify any case characteristics not encapsulated within the taxonomy's parameters.

Building on this concept, Szopinski, Schoormann, and Kundisch (2019) develop a systematic framework for conducting these taxonomy evaluations. They propose two fundamental questions: identifying who or what is being evaluated, and clarifying the object of the evaluation. Their framework, grounded in the larger context of the Framework for Evaluation in Design Science (FEDS) Research (Venable, Pries-Heje, & Baskerville, 2016), accommodates the evaluation of purely technical constructs as a separate category (Venable et al., 2016). The FEDS methodology unfolds in four progressive steps: (1) defining the goals of the evaluation, (2) deciding on the evaluation strategy, (3) determining the features to be evaluated, and (4) designing the individual evaluation sessions.

Contemporary blockchain projects identified in primary research are evaluated against the taxonomy in Table 5.4. Detailed exploration and discussion of these cases follows in Chapter 5.

Taxonomy Limitations

Taxonomic development relies on a literature review and scoping study and is fixed in time. Blockchain research continues to move quickly with articles, conference publications and

journals being added to the knowledge pool. Good taxonomies are extendable well into the future without the addition of new dimensions, unfortunately the researcher is limited in time and so must trust their process.

In summary, a taxonomy is employed in this study to systematically categorise and analyse blockchain consensus methods, thereby gaining an understanding of the characteristics and interrelationships. It provides a coherent framework for dissecting the domain of blockchain consensus methods with the goal of uncovering the base level design characteristics of different consensus methods.

4.3.2 Exploratory Factor Analysis

Hypothesis 2a (Figure 4.3) states that factors contributing to health in OSS can be identified through publicly available data. These factors are informed by the definition of health from Section 3.2.2: engagement, interest, and robustness as determined by a concept mapping in Table 3.1. There is no direct measure of what constitutes a construct such as engagement and so this can be modelled by using a latent factor composed of known metrics.

Exploratory Factor Analysis (EFA) is a multivariate statistical technique used for determining underlying constructs that are present in a dataset composed of a large number of variables. The constructs, or factors, can represent groupings within the data that are hypothesised, or known, but not directly observable. This unveils structure within the dataset and is often called *Factor Analysis* outside of the social sciences (Clark, 2018). *Latent factors* and *latent variables* are terms used interchangeably and represent inherent characteristics that do not have a well known associated metric[1]. EFA is common in fields like psychology and economics where participants are given a survey and the results are analysed (Finch, 2020b) but there is little work applying the technique to software engineering. OSS is a human lead directive combining social coordination with technical innovation, and as a socio-technical field is fit for application of these techniques to capture inherent structure that can define, for example, engagement.

In the present context this is the idea of *Sustainability*, *Robustness*, and *Niche Fit* (Sections 3.3 to 3.4, respectively) as applied to an open source software project, and the large number of variables are the possible set of metrics identified in Table 3.2, Table 3.3, and Table 3.4 respectively.

1. Linguistically, to keep with the theme of exploratory factor analysis, the term *factor* is used in this work with *variable* reserved for the indicators.

Figure 4.4 shows a factor diagram with the conventions of indicator (observed) variables, x_i, in boxes, latent variables (also factors or constructs) in ellipses, η_i, factor loadings, λ_i, and residual terms, ϵ_i. The unidirectional arrows show a predictive relationship from factor to indicator: $\eta \rightarrow x$. The bidirectional arrows are covariance between variables or factors, cov_{mn}, and factor variance, var_i.

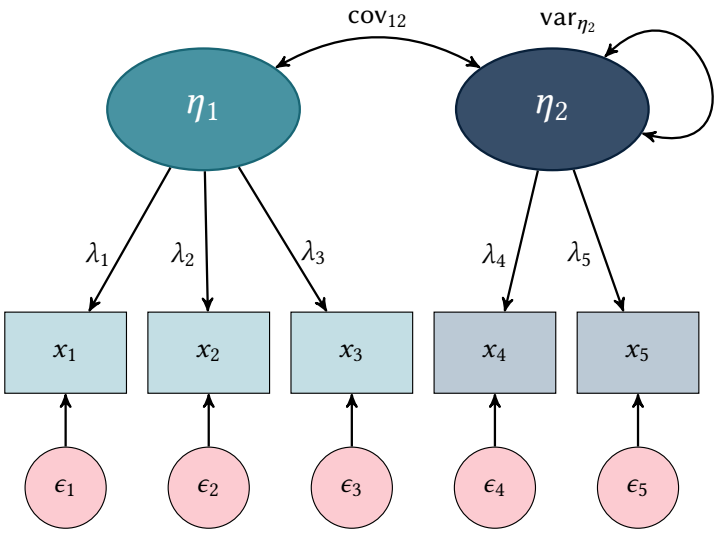


FIGURE 4.4: Factor analysis diagram showing two latent factors, five indicator variables, a covariance, factor variance and residual error terms.

EFA assumes that the latent construct (for example, *Robustness*) is responsible for the correlation of the indicator variables. In practice this allows the researcher to conclude on a statement of influence, that is: 'developer engagement is positively related to pull requests.' A double-headed arrow represents correlation and can be between latent factors, or indicator variables. EFA does not assume perfect measurement of observed variables and allows the factor to explain what the indicators have in common, and what is not held in common is due to measurement error (Clark, 2018). This error is shown in Figure 4.4 as a unidirectional arrow influencing the observed variable and includes everything responsible for the variance *except* the latent factor.

There are expected to be more than one latent factors in a dataset, up to a redundant but mathematically possible n factors in a dataset of n observed variables. The goal of an EFA is to determine an optimum number of factors that can adequately describe the characteristics of the underlying dataset while maintaining inherent structure. Figure 4.5 shows the six-stage EFA process which is employed in Chapter 6.

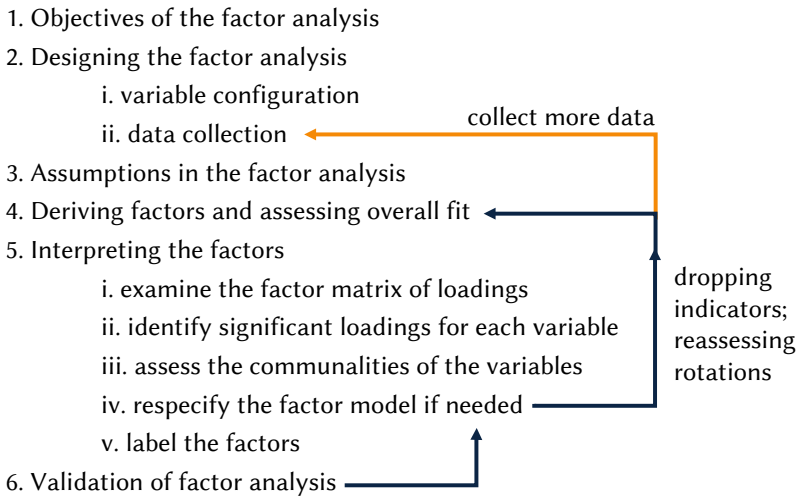


FIGURE 4.5: Six-stage process for conducting the exploratory factor analysis showing the data collection step and the iterative pathways taken to refine the model. Modified from Hair Jr. et al. (2014, pp.95–104).

EFA Comparison with PCA

EFA is similar in style but different both statistically and philosophically to another multivariate method, principle component analysis (PCA) and so deserves a brief mention. PCA operates by standardising the variance (diagonal of the correlation matrix) to unity, thereby considering all variance, inclusive of each variable's self-correlation (Hair Jr. et al., 2014). Its core objective is data reduction, where it attempts to distil the essence of numerous variables into a smaller set of principal components that captures the most variance.

EFA, conversely, is driven by the quest to discern latent constructs that are reflected in observed measurements. Although it can be employed for data reduction, EFA stands out, especially when the research agenda leans towards theory-building and unveiling underlying structures (Fabrigar & Wegener, 2012) as is the case in the present work. EFA is designed to explore potential underlying factor structures without imposing a specific structure on the outcome. It seeks to understand if there are any latent constructs that can explain the patterns of correlations among variables. In this sense, EFA does not begin with any assumptions about which variables might be redundant or unnecessary.

EFA Justification

EFA has proven to be a valuable tool in the fields of software engineering and IS; from human-computer interaction (Howard, 2016) to determining success factors for software application

integration (Gericke, Klesse, Winter, & Wortmann, 2010). Chandrachooodan and Radhika (2022) utilise EFA to develop a project management adoption methodology, identifying the essential factors that influence the successful implementation of project management practices in software engineering projects. Success appears as a theme in EFA studies, for example, What factors contribute to project success or failure? (Alcaraz-Corona, Mata, & Torres-Castillo, 2019); What success factors are attributable to sustainable management in IT service projects? (Zaleski & Michalski, 2021).

These studies demonstrate the relevance and applicability of EFA in the domains of software engineering and IS.

EFA Time Horizon

The time horizon for the EFA study is cross-sectional. The study is interested in determining the factors that contribute to engagement up to March, 2022. From this point, the historical data is collected going back as far as project data exists. Although a longitudinal perspective could provide insight into how projects evolve, the time constraints limit this avenue. More is discussed on this matter in Section 11.4.2.

EFA Data Collection

The research design stage answers how many, and what type of variables are to be included. Metrics are determined to adequately capture the characteristics of developer engagement, then data is collected and cleaned. Once the metrics are calculated, the dataset can be analysed. These steps are common in extracting and analysing open source data (Goeminne & Mens, 2013). Data for public blockchain projects are readily available for collection and analysis from GitHub through the web interface, programmatically through the API, and in raw archival form from the GHArchive. GitHub currently has 17 event types; all the events have author username and date metadata which can be used to develop further metrics.[2]

The raw data can be fed into a custom ClickHouse database.[3] ClickHouse is an open source column-oriented database management system designed for online analytical processing. This is ideal for large datasets that involve mostly read-only queries and batch updating.

Blockchain projects are identified using the CoinMarketCap API by ranking of market capitalisation as of March 2022. Details retrieved include project name, rank, website, and source code location if available. Section 6.3.3 has the full data story.

2. https://docs.github.com/en/developers/webhooks-and-events/events/github-event-types
3. https://clickhouse.com/

EFA Data Analysis

The data is analysed though factor analysis with the `psych` package in **R** statistical programming software. The analysis is presented in Chapter 6.

EFA Validation

Model validation is by two mechanisms: Confirmatory Factor Analysis (CFA) applied to the measurement model, and cross-validation by separation of the dataset into a training and testing segment. CFA is undertaken as part of the next method (Section 4.3.3), and thus cross-validation is most relevant here.

4.3.3 Structural Equation Modelling

Once indicator variable data is collected and factors identified using EFA, the structure between the factors is investigated. This structure is a directional relationship and leads to statements such as Hypothesis 2b (i) and (ii) in Figure 4.3: general interest generates developer engagement, and developer engagement leads to robust software.

To determine statistical significance and test these hypotheses, the method of Structural Equation Modelling (SEM) is used. SEM is a multivariate data analysis method that represents a fusion of factor analysis and multiple regression analysis techniques. It is used to determine relationships among latent constructs, which are unobserved theoretical concepts represented by several measurable variables known as indicators. SEM has the ability to simultaneously estimate multiple and interrelated dependence relationships, account for measurement error, and define a measurement model (Hair Jr. et al., 2014). The latent variables embedded in SEM serve to enhance the representation of theoretical concepts and consequently, improve the statistical estimation of the relationships between these concepts by adjusting for measurement error.

There are two prominent types of SEM: covariance-based SEM (CB-SEM) and partial least squares SEM (PLS-SEM). CB-SEM is characterised by estimating model parameters to minimise the discrepancy between the estimated and sample covariance matrices. PLS-SEM, on the other hand, seeks to maximise the explained variance of endogenous latent variables by estimating partial model relationships through an iterative sequence of ordinary least squares regressions (Hair, Hollingsworth, Randolph, & Chong, 2017).

SEM analyses are underpinned by a theoretical base. The theoretical grounding facilitates the establishment of causality and lays the groundwork for creating a meaningful model. This study uses CB-SEM as its primary goal is theory testing and confirmation (Hair Jr. et al., 2021).

SEM Justification

SEM is widely adopted in social science domains such as education (Khine, 2013) and psychology (MacCallum & Austin, 2000). In the context of IS, SEM's application extends to knowledge management (Cepeda-Carrion, Cegarra-Navarro, & Cillo, 2018), management information systems (Hair et al., 2017), supply chain management (Kaufmann & Gaeckler, 2015), and operations management (Bayonne, Marin-Garcia, & Alfalla-Luque, 2020). By 2010, SEM of the covariance-based variety emerges "especially well suited to research in information systems" (Dow, Wong, Jackson, & Leitch, 2008) and its use "widespread" (Roldán & Sánchez-Franco, 2012, p.194).

Narrowing the focus to software engineering reveals a paucity of SEM application. A 2021 survey of SEM using PLS finds only 29 studies since 2005 (Russo & Stol, 2021). Moreover, no more than two studies per year since 2015 are documented, and even less when restricting the methodology to CB-SEM suggesting an untapped opportunity to apply this methodology in the current domain.

Despite the relative novelty of employing SEM within the software engineering field, an increasing number of researchers begin to incorporate organisational and human factors into their studies. According to Russo and Stol (2021), the majority of these studies (41%) deal with aspects of professional software engineering practice, such as developer retention in open source software (Barcomb, Stol, Riehle, & Fitzgerald, 2019). Others engage with Agile development practices (Vijayasarathy & Turk, 2012; Campanelli, Camilo, & Parreiras, 2018), examine how user influence and user responsibility affect IS project performance (C. C. Chen, Liu, & Chen, 2011), or propose measurement models to quantify process harmonisation levels within organisations (Romero, Dijkman, Grefen, Weele, & Jong, 2015).

Only a handful of studies, however, apply SEM's potential for analysis into version control software. Chengalur-Smith et al. (2010) look into the sustainability of OSS across multiple time frames; Abdulhassan Alshomali (2018) models trends in GitHub programming languages via SEM, while Schroer and Hertel (2009) deploy SEM to dissect the structure of engagement tasks undertaken by Wikipedia volunteers using partial least squares path analysis.

These studies attest to the efficacy and validity of using statistical analysis tools like SEM for data collected from open source software. The prospect of employing SEM to assess software health, in particular, is a guiding motivation for the present research.

SEM Time Horizon

The time horizon for the SEM study is cross-sectional (the same as EFA, see Page 86) to determine the model structure for software health up to March, 2022.

SEM Data Collection

The same dataset used for the EFA from Section 4.3.2 is used for the SEM.

SEM Data Analysis

Data analysis of the structural model is carried out using the `lavaan` statistical software package in **R**. The analysis is presented in Chapter 8.

SEM Validation

Validation is by two mechanisms: first, CFA is a testing procedure to determine how well the measured variables represent the constructs in the model. This determines the validity of the measurement model which is the first part of the overall SEM. Secondly, measures of model fit as applied to the structural model including absolute measures and incremental fit measures (Hair Jr. et al., 2014).

SEM Limitations

It must be noted that "factor analysis will always produce factors" and is agnostic to "garbage in, garbage out" (Hair Jr. et al., 2014). Therefore, the role of the researcher is to ensure appropriate indicators are chosen, data is carefully collected, and factor results are thoughtfully interpreted. The factors produced are only as good as their descriptive power within a set context.

Sample size is another limitation, the present study meets the minimum requirements for EFA and SEM, however more (quality) data could bolster results and yield more generalisable insights.

Cross validation is necessary to avoid the situation where the model ends up being overfit to the data, affecting generalisability. Constraints in the data collection process on the number of available projects limit collecting an entire new dataset to validate against the original model. When splitting the existing dataset into two, sample sizes fall below recommended thresholds ($\approx$200) which in turn can affect fit statistics. It is important to be aware of this both when celebrating testing-fit and when lamenting testing lack-of-fit.

The thresholds commonly recommended for fit statistics used to verify the models are developed in the context of normally distributed data (Finch, 2020b), and must not be considered law. Acceptance or rejection of models should not be based on fit statistics, rather on the ability of the model to provide structure to the data.

4.3.4 Framework Development

A theoretical framework, in this context of IS, serves as a conceptual structure or model that organises and guides the understanding of a particular phenomenon, in this case, the health of software systems. It provides researchers with a systematic approach to analyse complex issues, facilitating the generation of meaningful insights and actionable recommendations.

The third research question of this thesis seeks to identify the components of a comprehensive model of blockchain software health. This question is guided by the third hypothesis, which posits that a well-constructed model of blockchain software health can stimulate innovation in blockchain technology (see Figure 4.3). In other words, the health of an open source software project is considered a crucial determinant of its capacity for innovation, with performance being a consequential effect of this health.

The primary artefact of this thesis is a framework for analysing software health, specifically within the context of blockchain technology. This framework serves as a tool for researchers and developers alike, enabling them to evaluate and enhance the health of their software projects, thereby driving innovation and performance improvements.

The field of software engineering faces a notable deficiency in the development of robust theoretical frameworks (Sjøberg, Dybå, Anda, & Hannay, 2008). Empirically-grounded theories within the domain are still at a nascent stage of development with the potential result of slowing the aggregation of knowledge within the discipline. While SE does possess implicit theoretical underpinnings, these theoretical constructs often lack widely recognised names and open discourse suggesting the need for a more formalised and openly debated theoretical framework within the field (Johnson, Ekstedt, & Jacobson, 2012).

The development and application of this framework tests the third hypothesis, contributing to the broader discourse on software health and its role in technological advancement.

Framework Justification

A framework is an artefact "representing both a model and a closely interrelated method to use [and] implement the model" (Kotzé et al., 2015). The contribution to knowledge emerges from the creation activity and is assisted through process, specifically on artefact instantiation (Vaishnavi et al., 2004). The framework as a tool is well known and can be seen used for analysis of open source data (Goeminne & Mens, 2013), taxonomical development (Nickerson et al., 2013) and assessment (Szopinski et al., 2019), and scoping study design (Arksey & O'Malley, 2005). Within framework development itself, frameworks are in place for theory in IS (Gregor, 2006; Beck et al., 2013), and methodological development in IS (Iivari, Hirschheim, & Klein, 2000). Hevner et al. (2004) lists theories and frameworks as foundations of the IS

knowledge base. A framework is a prime contribution of this work.

Framework Method

A framework for a high-level conceptual analysis of blockchain health is developed based on Gregor's seven structural components of theory (Gregor, 2006). This method is shown effective to create frameworks for artificial intelligence research (Bawack, Wamba, & Carillo, 2021) and IoT (Nord, Koohang, & Paliszkiewicz, 2019), and within the blockchain field the impact on supply chains (Treiblmaier, 2018), and e-commerce (Treiblmaier & Sillaber, 2021). The framework methodology is outlined in Table 4.1. The goal of the framework is double-sided, that is a combination of explanation and prediction. Explanation is the description of how and why with the intent of further developing the reader's understanding. Prediction is using the knowledge to define a future trajectory. This combination of explanatory and predictive theory is common in IS research (Gregor, 2006).

Table 4.1: The blockchain health framework involves seven components that when combined represent a new theory.

Theory Component	Description and Example
1. Means of representation	How the theory is visually or textually depicted. Example: tables, words, diagrams.
2. Primary constructs	Fundamental concepts or building blocks of the theory. Example: developers, metrics, popularity, engagement.
3. Statements of relationship	Connections or links between different parts of the theory. Example: Interest positively impacts developer engagement.
4. Scope	Defines the boundaries or extent of the theory. Example: Applicable to OSS.
5. Causal explanations	Reasons for why certain events or phenomena occur in the theory. Example: Robustness is a result of developer engagement.
6. Testable propositions	Statements that can be verified or falsified based on empirical evidence. Example: An increase in interest results in a corresponding increase in developer engagement.
7. Prescriptive statement	Offers guidance or recommendations based on the theory. Example: To strengthen developer engagement: encourage and facilitate pull requests.

These elements of theory closely match those of Sjøberg et al. (2008) who offer a framework for theory building in software engineering composed of defining constructs and propo-

sitions, providing explanations, determining the scope, and testing. Gregor's method is more robust and written in the context of IS and thus is the approach taken here. The theory development and framework components are further detailed in Chapter 9.

Framework Evaluation

Despite the significant emphasis placed on theory, the IS discipline experiences a gap in the advancement and honing of theoretical frameworks (Weber, 2012). An evaluation methodology is taken to ensure rigour in the development and application by consideration of the attributes of the theory's parts and attributes of the theory as a whole (Weber, 2012).

The framework must include testable propositions (Component 6 in Table 4.1) but also be falsifiable such that exclusive criteria narrow the scope of applicability (Gregor, 2006). Section 9.3.6 proposed statements that can be verified or falsified. Empirically, the framework is validated through the structural relationships between factors, as discussed in Section 8.7. The assessment of the framework also encompasses feedback and insights garnered from discussions surrounding the current work. Ultimately, the community's long-term appraisal determines the framework's efficacy and utility (Sjøberg et al., 2008). Over time, community members discern whether the framework is apt and beneficial or requires amendments. In the shorter term, the theory is evaluated in Section 9.4 by using a framework for evaluation within IS that consists of five criteria: level, importance, novelty, parsimony, and falsifiability (Weber, 2012).

Framework Development Limitations

The framework methodology, while offering a structured approach to analysing software health, is not without its limitations. One of the primary constraints is the specificity of cases. The framework, developed based on a comprehensive review of existing literature and empirical data, may not encompass all possible scenarios or contexts. Consequently, it may not fully capture the nuances of specific cases or be universally applicable to all blockchain projects. Additionally, the interpretation of the framework's results requires expertise and understanding of the blockchain field, and any misinterpretation could lead to incorrect conclusions or decisions.

Another limitation lies in establishing causality. Although the framework aims to identify relationships between constructs, proving that one factor directly causes another is often complex and requires further investigation. The level of generality of the framework may also pose a limitation. While it aims to be applicable to a wide range of scenarios, it may not capture the specific details of every individual case, potentially limiting its predictive power

in certain contexts. Lastly, the rapid evolution of blockchain technology could outpace the framework's ability to adapt, potentially reducing its relevance over time. These limitations should be considered when applying the framework to ensure accurate interpretation of its results and appropriate use of its predictions.

4.4 Research Design Summary

In this chapter, the philosophical research design and assumptions are outlined and the DSRM approach in IS is identified as the project's cornerstone methodology. This approach is a staple in IS research, allowing for multiple research entry points and allowance for iteration.

A literature review incorporating a scoping study informs the research questions, associated hypotheses, and testing techniques, and are all mapped in Figure 4.3. The initial research question explores the factors that contribute to consensus methods, operationalised through the development of a taxonomy.

The subsequent research question seeks to distil a definition of software health through an Exploratory Factor Analysis and mining of OSS blockchain repositories. Following the identification of the health factors, the next phase applies structural equation modelling to uncover the interrelationships between various health factors.

Lastly, a theoretical framework for evaluating open source blockchain health is constructed, aiming to illuminate areas for improvement and foster innovation, particularly within blockchain performance.

Next, Chapter 5 looks at the results of testing the hypotheses to answer Research Question 1: What are the factors that influence blockchain consensus?

Chapter 5

A Taxonomy of Blockchain Consensus Methods

> *The trouble with taxonomic boxes is... that they tend to be empty, however beautiful they are on the outside.*
>
> Kenneth B. Boulding
> *Illustrating Economics: Beasts, Ballads, and Aphorisms*
> 1980

Chapter 2 Provides an Overview of blockchain systems to highlight the research gap—lack of an adequate explanation of blockchain performance. As captured by the blockchain trilemma in Figure 2.8, scaling presents a bottleneck to the overall performance of blockchain systems. In this chapter Research Question 1 and the related Hypothesis are tested by a taxonomical classification of consensus methods. The methodology is described in Section 4.3.1.

5.1 Introduction

Research Question 1 asks what are the factors that influence blockchain consensus? From the literature Hypothesis 1 states there are distinct categories of consensus mechanisms, each with unique characteristics and scalability implications.

A taxonomy, in this context, serves as a systematic framework that categorises consensus mechanisms based on their distinct characteristics and scalability implications, facilitating a nuanced understanding of the factors influencing blockchain consensus. By distinguishing and systematically grouping these mechanisms, the taxonomy provides a structured lens to analyse the distinctive features and scalability implications.

This in turn aids in the empirical testing of the hypothesis. If the taxonomy reflects distinct categories of consensus mechanisms with unique scalability implications, it supports the hypothesis. Conversely, any observed deviations guide further refinement of the understanding.

Blockchain consensus is the continuous process of the network proposing updates and updating the chain state, as reviewed in Section 2.3.1. A taxonomy of the different blockchain consensus methods is completed.[1] Nine surveys meet the criteria and are reviewed to extract the approaches to blockchain consensus found in the literature. In summary: sixty-nine unique blockchains are found and catalogued in Table 5.1 yielding the taxonomy in Table 5.3.

5.2 Taxonomy Methodology

Creating a taxonomy is a multi-stage process. First the literature is collected and assessed in a process similar to a literature review. Present taxonomies in the field of blockchain consensus are identified to ensure a meaningful contribution can be made to the field. The final stage applies the method by Nickerson et al. (2013) for developing the taxonomy. This is a well-known method within IS research (Section 4.3.1) for designing and refining a taxonomy (Szopinski et al., 2019) and begins with identifying a meta-characteristic, then defining stop conditions,

1. The findings in this chapter are published in *Cryptography* (Nijsse & Litchfield, 2020).

and lastly iterating to build the taxonomy. The iteration adds characteristics to be grouped into dimensions until the stop condition is satisfied in an inductive manner (Bailey, 1994).

Identifying the Relevant Studies

Gathering from a wide range of source material is considered because with blockchain research much of the information comes from grey sources such as industrial white papers, blog posts, and technical reports. There is a natural time-lag between business projects utilizing new open source technical details and academic literature that has study and review similar projects. This gap is beginning to close for blockchain; in 2012 there are only twenty-two technical reports published about Bitcoin (Decker, 2020), and by 2018 there are hundreds including peer-reviewed journal articles and surveys.

The date range for this study begins in 2009 with the introduction of Bitcoin; any results before this time are excluded. Electronic databases are the primary source of peer-reviewed literature and include: Institute of Electrical and Electronics Engineers (IEEE) Xplore, Association for Computing Machinery (ACM), Association for Information Systems (AIS), MathSciNet, SpringerLink, Scopus, and ScienceDirect. Two important sources that include grey literature stand out in the field of cryptocurrency: the Comprehensive Academic Bitcoin Research Archive (CABRA) (Decker, 2020), and Apograf—a database of papers specifically for distributed computing, cryptography, and blockchain related technology (Research Centre Holding Ltd, 2020).[2] Lastly, the top results from Google Scholar and the arXiv[3] preprint repository from Cornell University are also browsed to round out the search.

The search included studies that are reviewed in a secondary source, plus `blockchain` AND `consensus` AND/OR `decentralised` AND/OR `database` AND/OR `system`. From the found set, the following exclusion criteria are applied:

- Papers focussing on blockchain applications including, but not limited to finance and IoT as these do not analyse methods themselves, rather their applicability to the chosen application;
- Papers that do not address blockchain consensus or are inconsistent;
- Papers proposing new algorithms (primary research).

2. Apograf is no longer active as of 2021.
3. https://arxiv.org/

5.2.1 Surveys of Consensus Methods

Many have compared and contrasted different consensus methods however few have produced a comprehensive survey. The most recent to date is by W. Wang et al. (2019). The authors simplify consensus into PoX (all proof-of type methods) and other concepts, or virtual methods. Xiao et al. (2020) offer a five-component framework for blockchain consensus consisting of information propagation, incentivisation, block proposal, validation, and finalisation. The later three components – proposal, validation, and finalisation – represent exactly what consensus must accomplish, whereas incentivisation is an important aspect but not directly relevant to a consensus algorithm. Bano, Sonnino, et al. (2017) draw a split between proof-of-work and PoX overlapping with a third Venn intersection called hybrid models.

Dinh et al. (2017) develop a framework for analysing private blockchains and in doing so evaluate ten different blockchains. Additionally the prominent enterprise blockchains are surveyed with respect to their fault models by Cachin and Vukolić (2017). An energy savings attribute is presented by Chalaemwongwan and Kurutach (2018), although somewhat arbitrarily. Subsequent surveys by Bach, Mihaljevic, and Zagar (2018), Chaudhry and Yousaf (2018), and Tasca and Tessone (2019) continue in this vein, selecting and comparing blockchains that are sufficiently different.

Nine surveys are use as a basis for the taxonomy. Some of the general themes on how classification schemes may be derived emerge in the papers: the grouping of consensus methods as PoX versus any other concept such as virtual methods and hybrid models (W. Wang et al., 2019; Bano, Sonnino, et al., 2017); a framework for blockchain consensus consisting of information propagation, incentivisation, block proposal, validation, and finalisation (Xiao et al., 2020), where proposal, validation, and finalisation are the steps for consensus but incentivisation is not relevant to the consensus algorithm; comparing the functionalities provided in private blockchains (Dinh et al., 2018) or by comparing fault models (Cachin & Vukolić, 2017); other means for classification include topics criticising blockchains, such as energy usage (Chalaemwongwan & Kurutach, 2018); identifying differences between specific blockchains rather than protocols (Bach et al., 2018; Chaudhry & Yousaf, 2018; Tasca & Tessone, 2019).

The selected surveys provide 69 consensus methods as empirical data points in the taxonomy. Table 5.1 lists the blockchains by name (for example, Hyperledger); however, this could also refer to the protocol maintaining consensus (for example, BFT SMaRt), or the name of the company (for example, Ripple). The lack of a naming convention is confusing for researchers looking for a concise summary, for example, Hyperledger is mentioned in six of the surveys but it is not clear what variant is referred to. Hyperledger is an umbrella suite of products and offers support for different consensus mechanisms, including BFT SMaRt and PBFT (Fabric),

TABLE 5.1: Blockchain projects surveyed across nine studies and sorted according to the consensus family the protocol is derived from. PoW spawns the most number of derivatives, followed by PoS, and PBFT.

	Consensus	Source Study								
Name	**Family**	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
PoW (Bitcoin)	PoW	•	•	•	•	•	•	•	•	•
Bitcoin NG	PoW		•						•	•
PoBurn	PoW				•				•	
Decor + hop	PoW		•							
Ghost	PoW		•							
Scratch-off puzzles	PoW							•		
PoParticipation and Fees	PoW						•			
Spectre	PoW		•							
PoPublication	PoW				•					
PeerCensus	PoW [1]								•	•
ColorCoin	PoW [1]							•		
Counterparty	PoW [1]							•		
Hyperspace	PoW [1]							•		
Multichain	PoW [1]	•								
NameCoin	PoW [1]							•		
OmniLayer	PoW [1]							•		
Po eXercise	PoW [2]					•			•	
PoUseful Work	PoW [2]								•	
PoStake (Ethereum)	PoS		•	•	•	•	•	•	•	•
Algorand	PoS		•						•	•
Ouroboros (+ Praos)	PoS		•			•				•
PoActivity	PoS				•				•	•
Snow White	PoS		•						•	•
Casper	PoS								•	•
NXT (Ardor)	PoS							•		•
Chain of Activity	PoS									•
PoStake Velocity	PoS								•	
Hyperledger	PBFT	•	•	•	•			•		•
Implicit Consensus	PBFT						•			
Iroha	PBFT [3]	•								
Kadena (Juno)	PBFT [4]	•		•						
Honeybadger	PBFT [5]	•	•							•
Hybrid Consensus	PBFT + PoW								•	
Omni Ledger	PBFT + PoW		•							
ByzCoin	PBFT + PoW [6]		•							•
Solidus	PBFT + PoW [6]		•							
Elastico	PBFT [7]		•				•			
Chainspace	PBFT [8]		•							

Continues on the next page →

TABLE 5.1 Blockchain consensus methods sorted by family – continued from the previous page

Name	**Family**	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
Ripple	FBA	•		•	•	•	•	•		•
Stellar	FBA	•				•		•		•
Chain	FBA	•								
DelegatedPoStake	DPoS				•	•		•	•	•
PoAuthority	DPoS			•					•	•
PoImportance	DPoS				•	•				
Dfinity	DPoS			•						
PoVote	DPoS						•			
Sawtooth Lake	PoET	•	•	•	•				•	•
PoLuck	PoET					•			•	
Resource Efficient Mining	PoET		•						•	
PoOwnership	PoET								•	
Raft	Raft	•		•						
PoTrust	Raft						•			
Quorum (JPMorgan)	Raft [9]	•								
Tendermint	BFT	•		•	•			•	•	•
Cosmos	BFT							•		
Corda (& Enterprise)	BFT-SMaRt+Raft	•						•		
BFT SMaRt	BFT SMaRt	•								•
Symbiont Assembly	BFT SMaRt	•								
PoCapacity	PoCapacity		•		•			•		•
IOTA	Hash DAG	•								•
Hashgraph (Swirlds)	Hash DAG	•								
Paxos	Paxos	•								
PoHumanWork	PoBiometrics								•	
PoMemory	PoMemory								•	
Ethash	PoMemory								•	
PoSpace	PoSpace		•		•			•	•	•
Filecoin	PoSpaceTime								•	
Peercoin	PoS + Coin Age							•		•
RSCoin	2PC		•							

Source studies: [1] Cachin and Vukolić (2017); [2] Bano, Sonnino, et al. (2017); [3] Dinh et al. (2018); [4] Chalaemwongwan and Kurutach (2018); [5] Bach et al. (2018); [6] Chaudhry and Yousaf (2018); [7] Tasca and Tessone (2019); [8] W. Wang et al. (2019); [9] Xiao et al. (2020).

Notes: [1]based on Bitcoin, [2]useful, [3]based on Bchain, [4]Scalable BFT, [5]asynchronous, [6]with a committee, [7]with sharding, [8]flexible, [9]Istanbul BFT.

Acronyms: **Po** Proof-of—; **PoET** Proof of Elapsed Time; **PoW** Proof-of-Work ; **BFT** Byzantine Fault Tolerant; **PoS** Proof-of-Stake ; **SMaRt** State Machine Replication ; **PBFT** Practical Byzantine Fault Tolerant; **DAG** Directed Acyclic Graph ; **FBA** Federated Byzantine Agreement ; **2PC** Two-Phase Commit; **DPoS** Delegated Proof-of-Stake.

PoW and PoAuthority (Besu), Proof of Elapsed Time (PoET) and PBFT (Sawtooth). The table is sorted by the consensus family that the method is derived from, for example, PoW is the only method mentioned in every survey but the original version used in Bitcoin is just one of many alternatives.

5.2.2 Prior Work

Considering the volume of surveys that are written about blockchains there are only a few that attempt a taxonomy. One of the early classifications is from 2015 and although called a taxonomy of consensus systems the authors concentrate on blockchain applications rather than the protocol to maintain the ledger (Glaser & Bezzenberger, 2015).

A review of papers that present taxonomies of blockchain consensus protocols shows some common trends. The approaches reviewed are thorough but including everything into a taxonomy, as most try to do, is ultimately unwieldy and fails to include additional elements that may be blockchain-specific but that provide nuance. With the exception of one that categorises blockchain applications rather than the protocol to maintain the ledger (Glaser & Bezzenberger, 2015), most include consensus as just one category. For example, providing a taxonomy from a systems-architecture viewpoint where consensus is one dimension among others, so a systems designer can choose an appropriate blockchain style (Xu et al., 2017). Two taxonomies attempt to categorise all the components of a blockchain (Tasca & Tessone, 2019; Wieninger, Schuh, & Fischer, 2019) and consensus is included as a dimension among others, like the open source nature of the codebase and the financial classification of the token. A comprehensive survey is found in W. Wang et al. (2019) but offers no formal classification and has some confusing categories. The authors delineate consensus into four categories: permissionless consensus, PoW-style, proof-of-resources, and proof-of-concept for performance improvement, while categorising PoS separately. The last survey provides a first-principles approach by creating a classification tree structure with the underlying network assumptions at the root so that point-to-point methods among authenticated channels are separate from the p2p network (J. A. Garay & Kiayias, 2019). In this case, the taxonomy tree illustrates that consensus is branched with public/private setup, computational assumptions, and communication cost. So while examples of PoS in the p2p setting are provided, a fine-grained distinction between blockchain protocols is not present.

In general terms, a taxonomy is an attempt to gain understanding of a group of instances by grouping and categorising apparent features or characteristics and creating an abstract model that may be used to compare with newly found instances. The characteristics, when categorised, allow for the identification of meta-characteristics from which the model is comprised. A new instance, when compared to existing instances, can then be considered in

respect of what is already known and a taxonomy may be refined or developed as new knowledge is accrued. Thus, taxonomies tend to emerge once a researcher has sufficient experience with a field to see subtle distinctions between characteristics.

5.3 Derivation of Categories

To develop the taxonomy, an iterative three-step approach is applied that applies design science (Nickerson et al., 2013). The apparent characteristics identified for the development of this taxonomy reflects the understanding that blockchain consensus is about maintaining the state of a ledger that is replicated across many nodes in a decentralised system. Implementation specifics are not included in the taxonomy.

Step 1: Derive the principal meta-characteristic for categorisation; the methodology for maintaining a distributed ledger at a high-level. Characteristics, similar to deterministic finality, or committee-based voting, are included in this category. The meta-characteristic is defined as a high-level methodology for maintaining state of a distributed ledger.

Step 2: Adopt an inductive approach that determines which characteristics may be grouped into dimensions empirically by identifying characteristics within the meta-characteristic and subsequently selecting dimensions to group them together (Bailey, 1994). The conceptual-to-empirical approach provides the opportunity to hypothesise about new dimensions that can be tested with data.

Step 3: Check whether there are characteristics left unresolved or uncategorised and iterate if necessary, back to either an empirical-to-conceptual or conceptual-to-empirical strategy.

Table 5.1 lists consensus methods that provide an empirical starting point for the taxonomy. The blockchains in Table 5.1 are sorted by the number of occurrences in literature to derive a prominence ranking (Table 5.2). While the number of mentions is a crude means for deriving prominence, it does afford some degree of conscious acknowledgement of importance in the community. Nineteen consensus categories are identified and provide a basis for the first iteration. The taxonomy incorporates the methods from Table 5.2.

As a feature, PoW is the key to solving the double-spending problem for a cryptocurrency by incentivising the longest-chain rule. Thus, it represents the first dimension in the taxonomy and signifies a finite resource; that is, a limited resource from which value may be apportioned. The characteristics of this dimension can be determined directly from Table 5.2. For example, in proof-of-stake, the stake represents the proportion of total tokens that a node has dedicated to consensus, making *tokens* the characteristic associated with it.

Identifying a new dimension requires at least one iteration. The approach applied involves starting with a concept and drilling down to derive empirical outcomes. In one survey, a dis-

TABLE 5.2: Consensus methods ranked by occurrence in literature. For the acronyms refer to Table 5.1.

Consensus Method	Occurrences	Method (Con't)	Occurrences
PoW	33	BFT SMaRt	4
PoS	27	PoCapacity	4
PBFT	20	Hash DAG	3
FBA	13	Honeybadger	3
DPoS	12	PoCoinAge	2
PoET	11	PoMemory	2
Raft	8	2PC	1
Tendermint (BFT)	7	Paxos	1
PoSpace	5	PoBiometrics	1
		PoSpaceTime	1

tinction is made between networks operating in an open p2p manner and those that require trusted channels (J. A. Garay & Kiayias, 2019). Therefore, a dimension for *network communication* is included, with the characteristics of *trusted channels* and *p2p*.

5.4 Taxonomical Dimensions

Consolidation of the blockchain surveys and the taxonomy reviews above results in seven dimensions that are then delineated by their characteristics. Six unique resources are identified: clock-cycles, bits, tokens, votes, time, and biometrics. The blockchains are further differentiated by the dimensions of fault tolerance, block proposal mechanism, transaction finality, network timing assumptions, accessibility and communication.

Resource Scarcity

Clock-cycles The *work* in a proof-of-work system is the computational work done by the processor in finding a hash value subject to some target requirement. Similar to physics work being non-reversible, the computational work done by a chip cannot be recovered or undone. Clock-cycles are scarce because the energy input has non-zero cost. A cryptographic requirement here is that the hash function is one-way. Given any hash output, the data is easily verifiable and cannot be inferred.

Tokens Proof-of-stake removes the thermodynamic inefficiency in flipping bits by allocating users a stake in the system, proportional to their tokens. This stake can then be used to scale incentives for users that participate, generally by committing their tokens in a manner that could result in negative consequences.

Votes A BFT system must determine consensus by the replicas (nodes) voting on the state. Votes are scarce because a replica is only permitted one vote per round, however the votes have no value, in subsequent rounds all replicas are allocated a further vote. Classical consensus methods that maintain state in a non-blockchain system such as Paxos use voting to elect leaders (see Section 2.1.2). Many of these methods are adopted for blockchain consensus and thus bring their votes-as-a-scarce-resource characteristic. Nearly half of the methods in Table 5.3 employ votes as the scarce resource which are different than staked tokens as they hold no utility other than determining a majority.

Bits represent the state of a transistor and occupy a finite amount of disc space. Proof of capacity regimes allocate a user's stake in the system by a proportion of disc space. Just as tokens that are staked, disc space cannot be used for other purposes if it is participating in the blockchain. Disc space scales more slowly compared to processing power and may reduce a blockchains potential to be dominated by ASICs.

Time is independent of computing advances. As clock-cycles and read-write times get faster, blockchains secured by these resources may be exposed to unforeseen factors such as accelerated hardware attacks or temporal vulnerabilities. Proof of elapsed time (PoET) are chips with separate execution environments, or enclaves, that are inaccessible to the system. These modules can return a random delay to a process that can be used to assign block proposers.

Biometrics are a range of indicators that can verify identity or life. Similar to a hash function requiring a known average number of clock-cycles, a blockchain based on proof-of-biometrics can require a unique biological solution. The Completely Automated Public Turing-test to Tell Computers and Humans Apart (CAPTCHA) system is easy for humans to solve while requiring some small amount of time is an example of using a biological system to solve a puzzle.[4] Although there is no current blockchain project incorporating biometrics, it is proposed by Blocki and Zhou (2016), and adds a biological element to the dimension of scarce resources.

4. Time here is implicit in the growth of the biological system.

TABLE 5.3: Taxonomy of blockchain consensus methods. The primary classification is by Scarce Resource that the blockchain occupies. †cc is clock-cycles; D is deterministic; P is probabilistic; *Ethereum switched from Proof-of-Work to Proof-of-Stake in September, 2022.

	Scarce Resource												
Consensus Method	bits	time	cc†	tokens	biology	votes	**Fault Tolerance**	**Transaction Finality**	**Network Timing**	**Block Proposal**	**Network Accessibility**	**Network Comm.**	**Example Blockchain**
PBFT						•	3f+1	D	partial	election	public/private	trusted	HL Sawtooth
BFT-SMaRt						•	3f+1	D	partial	election	private	trusted	R3 Corda
Federated BA						•	5f+1 – 3f+1	D	partial	committee	consortium	both	Stellar, Ripple
Honeybadger						•	3f+1	P	async.	random	public/private	both	POA Network
2PC						•	2c+1	D	async.	none	private	trusted	RSCoin
Paxos						•	2c+1	D	async.	election	private	trusted	none
Raft						•	2c+1	D	async.	election	private	trusted	Quorum
Hashgraph						•	3f+1	P	async.	none	private	p2p	Hashgraph, IOTA
DPoS				•		•	3f+1 – 2f+1	P	synch.	committee	all	trusted	Steem, EOS
PoS				•			3f+1	P	s,p	committee	all	trusted	Ethereum*, Decred
Tendermint				•			3f+1	D	partial	election	private	both	Cosmos
PoW			•				50%	P	synch.	random	public	p2p	Bitcoin
PoCoinAge		•		•			50%	P	synch.	random	public	p2p	Peercoin/Ppcoin
PoElapsedTime		•					2c+1	P	synch.	random	public	trusted	HL Sawtooth
PoRetrievability	•						50%	P	synch.	random	public	p2p	Permacoin (PoR)
PoSpace	•						50%	P	synch.	random	public	p2p	Spacemint
PoSpaceTime	•	•					50%	P	synch.	random	public	p2p	Filecoin
PoMemory	•		•				50%	P	synch.	random	public	p2p	Zcash, Beam
PoHumanWork		•			•		-	-	-	-	-	-	none

Fault Tolerance

At the base level, fault tolerance refers to crash-fault tolerance where a node can fail and will resume operation once it is brought back online. These nodes cannot exhibit arbitrary behaviour, such as sending faulty information and so if a single node becomes compromised, the entire system is compromised. Consensus requires a majority of nodes, for example, with 2PC up to c nodes can crash, requiring $2c + 1$ replicas. Systems, such as Google's Bigtable, are guaranteed five replicas and can tolerate two failures. If the replicas can be subverted and send incorrect information the system must be Byzantine-fault tolerant. PBFT and its derivatives can handle up to f Byzantine faults of $3f + 1$ replicas.

Above, f and c are integers; whereas for a proofing type method, fault tolerance is the percentage of total resource that may be sacrificed before consensus is lost. Peer-to-peer systems assume bad actors will attempt to subvert the network, possibly colluding with each other, and may only be held off for as long as there is an honest majority. A 51% attack occurs when adversaries obtain >50% control of the scarce resource and can then alter, control, predetermine, or direct the consensus process. The selfish mining strategies of Eyal and Sirer (2014) show that an actor with <50% can withhold blocks and earn more of the reward; however, this does not affect liveness or safety.

Block Proposal

The question of who gets to propose new blocks (Figure 2.7) is fundamental to blockchain consensus. Selecting a validator must be fair and secure. A decentralised open system allows any participant the opportunity to propose blocks and a fair way to accomplish this is by random selection. If peers do not find out who proposed the block until after it is proposed, this is a leader-free scenario. Leader election can also be accomplished by using randomised processes at which point the lead replica is responsible for coordinating the subsequent update. These systems are usually private as replicas require known IDs. A committee-based system relies on a predetermined set of validators to be responsible for updates. Participants may join the network but not necessarily be part of the committee.

The exception occurs in the case of the DAG, in which no node is responsible or chosen for updates and there is no block proposer. A DAG links transactions in a similar manner to a blockchain with the exception that a graph node can have n outgoing vertices. A traditional family tree satisfies this condition; however, so does a PoW chain in the presence of n forks (not necessarily in consensus). With IOTA, the graph breadth continues to grow.

Transaction Finality

There are two categories describing how a transaction gets settled on chain. The first is deterministic and guarantees the data are written and committed to the blockchain at some known point after the block is posted. The second is probabilistic, where a transaction is confirmed with increasing probability as more blocks are added. The probabilistic approach says there is a chance that transactions are added to a fork of the chain that does not represent the most computational work. Over time the longest chain will emerge and forked blocks become orphaned. Bitcoin's proof-of-work is often called *Nakamoto* or *emergent* consensus because the more chain work that is done after a transaction is in a block, the more likely it is to be committed to the ledger (J. A. Garay, Kiayias, & Leonardos, 2020). PoS systems, such as Ethereum's Casper (Buterin & Griffith, 2017) have checkpoints before which all data finality is probabilistic and subject to reorganisation. After a checkpoint is reached, the transactions are finalised.

Network Timing

The timing considerations may be synchronous, asynchronous, or partially synchronous. Networks such as PoW and proof-of-capacity are generally considered synchronous because they are guaranteed to update the state upon completion of every round. What is not guaranteed is that every round will complete because there are no timing assumptions; the protocol will run as long as necessary to append a block. Recently it is demonstrated that Nakamoto consensus, operating in a dynamic participant pool, maintains safety and liveness in bounded-delay networks (J. A. Garay et al., 2020). A fully asynchronous network has no known upper bound for message delivery. A limitation of 2PC is that it will stall if a message is delayed for an arbitrarily long amount of time (Fischer et al., 1985). A partially synchronous model may employ timeouts, rules, learning algorithms, and predetermined hierarchies to resolve deadlocks. See Section 2.1.1 for more on safety and liveness; Section 2.1.3 for more on synchrony.

Network Accessibility

Categories for blockchain network access are public, private or a combination of the two. In a public network, anyone can join the network and participate, then leave the network without penalty. Most decentralised blockchains are public and so they need to be secured against Byzantine behaviour. Private blockchains require nodes to be validated and external participants may not be able to participate or view activity. Enterprise blockchains are typically private for a range of reasons, including efficiency and security. A consortium is comprised of a group of parties, such as financial institutions that share access to a network. The reasons

for establishing a consortium vary and include common goals, sharing of resources, mutual agreement on consensus methods, development opportunities, and so on. Individual consensus methods, such as PoS, may have instantiations of different access types, for example, PoS may be public as in Decred, or consortium as in EOS. Any public PoS system can be adapted for private use.

Network Communication

Nodes exchange transaction or block data via a range of network communication methods. A trusted setup requires nodes to validate each other through a key-exchange procedure or similar. Open p2p networks communicate via a gossip protocol flooding nearest-neighbours with information until all nodes are in agreement. Network Communication could be both point-to-point and p2p, for example, trusted channels can be built on top of a p2p network (A. Miller, Xia, Croman, Shi, & Song, 2016).

5.5 Analysis

This taxonomy applies to researchers and analysts investigating the current state in blockchain consensus to classify various types and variations. Researchers may be looking to identify areas for future development or optimisation while analysts could be looking for a starting point in adopting a consensus method to use or to base a blockchain design on. While this classification system is derived from existing sources and represents a current review of types, it may also be subject to change and update as new methods or approaches emerge.

Three categories are identified in Table 5.3: (1) blockchains based on traditional consensus and dependent on replica voting; (2) the need to demonstrate that some resource is consumed, exploited, or set aside; (3) dependency on tokenised representation. All three categories have valid uses but there is no ideal method. Furthermore, to accommodate more users and activity, much research focusses on the use of distributed systems as a framework or means for scaling protocols.

PBFT, BFT SMaRt, and FBA are grouped together and have well known blockchains. Depending on client need, Hyperledger incorporates a number of consensus algorithms. R3 Corda can use BFT SMaRt, which is an optimised BFT algorithm that can achieve a high throughput (Bessani et al., 2014) while Hyperledger Sawtooth has a PBFT implementation. FBA has a federation of permissioned validators whereby each validator determines its own set of nodes to establish trust. The pool of validators in Ripple is called a unique node list and in Stellar a quorum slice.

Paxos, 2PC, and Raft are categorised together and are not well represented in this space due to the algorithms being crash-fault (not Byzantine) tolerant, and requiring a network with known participants. Of note, 2PC splits communication into a prepare phase and a commit phase. In the prepare phase nodes are made aware of the state update and the central coordinator tallies the votes. The commit phase involves another round trip whereby each node updates their ledger. Paxos itself has no known blockchain implementations and is included in the taxonomy as a reference point. Raft is found in Quorum by JPMorgan and is an exception.

PoCapacity schemes that monopolise disk storage are interesting because of their resource efficiency versus PoW and their application to distributed storage. An alternative seen in proof-of-retrievability requires that a participant verify they have stored a portion of a file for later use. This example could be applied to a large public dataset where storage provides a distributed archive, such as Permacoin. Alternatively, proof of space as proposed for Spacemint seeks to apply disk space in a mining-like capacity where the user dedicates a portion of disk space to a large file and can verify that they have done the computational work to pebble a DAG from the file (Park, Alwen, Fuchsbauer, Gazi, & Pietrzak, 2015). Lastly, Filecoin uses a time dimension in their proof of spacetime consensus method by taking into account an amortisation period. The prover must show a recursive proof of retrievability to show they have maintained the storage for a period of time (Benet & Greco, 2018).

The use of biometrics in providing evidence of a scarce resource holds promise because individual people are unique and that combined with time implies that a transaction cannot be replicated or accelerated by computing advances. However, time is difficult to verify computationally due to advances in hardware and parallel processing. PoET uses special enclaves in a chip architecture called trusted execution environments such as Intel's Software Guard eXtension and ARM's TrustZone. Hyperledger Sawtooth has a PoET implementation. Network accessibility in Table 5.3 is listed as public, but a node is required to have the specific hardware module to participate. A PoW blockchain can act as a proxy for proof of time as it is a linked list in the manner of secure time stamping although a good PoET implementation can be more energy efficient.

5.6 Case-Study Validation

The present taxonomy is developed with secondary research literature and has excluded papers proposing new blockchain mechanisms. To validate the taxonomy the dimensions should encapsulate novel methods. Five cases are considered and applied to the taxonomy: Thunderella, Avalanche, LibraBFT, Gemini, and Solana. Table 5.4 shows the cases; none of which

TABLE 5.4: Taxonomy validation results applied to five contemporary protocols not found in the secondary research.

Dimension	Avalanche	Thunderella	LibraBFT	Gemini	PoH†
Scarce Resource	votes	cc*; votes	votes	clock-cycles	stake; time
Fault Tolerance	$1-\epsilon$	25%-50%	$3f+1$	50%	$3f+1$
Block Proposal	none	election	committee	random	election
Finality	probabilistic	deterministic	deterministic	probabilistic	deterministic
Network Timing	synchronous	async., sync.	partially	sync.	sync.
Accessibility	public	public, private	consortium	public	public
Communication	p2p	p2p, trusted	trusted	trusted	p2p

* cc is clock-cycles
† PoH is Solana's Proof-of-History

are used in the development and refinement of the taxonomy.

Avalanche (Team Rocket, 2018) uses a probabilistic safety guarantee similar to proof-of-work but without the resource intensive dependence on PoW. The probability that safety will be upheld is set by a parameter, ϵ, and determined by the designer. Avalanche uses an append-only DAG to maintain the public ledger.

Thunderella (Pass & Shi, 2018) is a hybrid of classical asynchronous consensus methods with synchronous blockchain methods. It improves the performance of proof-of-work chains by using an optimistic path that can tolerate 25% Byzantine nodes in an asynchronous environment to allow for more throughput, and can fallback to rely on a more standard 50% BFT in a synchronous environment in case of trouble. A committee is selected for leader election progressing in rounds in a permissioned environment or by a proof-of-work oracle in a permissionless setup.

LibraBFT (Baudet et al., 2019) is a BFT consensus mechanism operating under a partial synchrony assumption that is the basis for the Libra blockchain. Developed by Facebook (Meta), this project intends to be a global payments system.[5] The state of the ledger is maintained in a proof-of-stake style system by a set of trusted validators that can tolerate one-third faulty nodes.

The Gemini dollar (Gemini Trust Company, 2019) is a cryptographic token pegged to the US dollar. Known as a stable-coin, these are useful for merchants to transact without the volatility of a cryptocurrency. The Gemini dollar is an application built on the Ethereum platform and therefore must adhere to the rules of the Ethereum consensus mechanism.

Proof of History (PoH) is the name of the consensus method of Solana (Yakovenko, 2017). Solana utilizes a unique combination of PoH and PoS as its consensus mechanism. The scarce

5. Meta's Libra project becomes Diem and is shut down with the intellectual property acquired by Silvergate Capital in January 2022 (De, 2022).

resources in this context are notably stake, represented by bonded tokens, and time, which is indirectly incorporated through the PoH. Regarding fault tolerance, Solana is generally perceived to have a tolerance of $\frac{n}{3}$, where n signifies the number of validators, aligning it with other BFT algorithms. Solana leverages a verifiable delay function (VDF), to select the leader for block proposals. As for transaction finality, Solana is engineered to achieve fast finality, often realizing it in under a second under optimal conditions, thus providing what is frequently considered deterministic finality due to the expedited transaction finalization. Solana can be classified as synchronous p2p, owing to its use of the PoH mechanism which creates a historical record, certifying that a given event has transpired at a specific point in time.

The taxonomy provides a concise picture by limiting itself to seven dimensions and concentrating on the meta-characteristic of maintaining the state of a distributed ledger. Lastly, it is extensible to accommodate future algorithms as demonstrated by the case studies that are considered and applied to the taxonomy.

5.7 Conclusion

The chapter presents a taxonomy in which blockchains are categorised by consensus family across seven dimensions: scarce resource, fault tolerance, block proposal mechanism, transaction finality, network timing assumptions, network accessibility, and network communication. The taxonomy provides a robust contribution to the field that includes 69 blockchains that are presented in peer-reviewed literature. While the taxonomy offers a high level explanatory view, it is concise because it is limited to seven dimensions and concentrates on the meta-characteristic of maintaining the state of a distributed ledger. Lastly, to accommodate future algorithms, the taxonomy is extensible and to demonstrate this, as well as to evaluate the taxonomy on cases that are not involved in its development, five case studies are applied.

Limitations that are present in the taxonomy are that it is a snapshot of the present state of consensus and while blockchain research is expanding, blockchain variants are proposed faster than they appear in academic sources. Examples of blockchain implementations are given for reference; however, this is not a complete listing nor does the taxonomy classify individual blockchains.

Opportunities for development and research present themselves as further in-depth analyses. For example, a large number of blockchains use some form of proofing method to attempt to publicly verify that a scarce resource is secured but others utilise BFT methods from distributed computing. At this point, there is no clear future direction where consensus will be focussed. In the meantime, other methods such as DAG with a gossip protocol, Solana's PoH,

or asynchronous BFT have yet to be tested at scale.

Additionally, biometrics and time-based approaches present opportunities in sociotechnical systems and offer interesting areas for future development. For example, social status or reputation provides a strong incentive to maintain integrity within a network by increasing social scores for good behaviours. Attempts to subvert the system are negatively reinforced by penalizing the social score, which takes time to accrue.

Next, in Chapter 6 the factors that can determine a picture of OSS blockchain health are explored.

Chapter 6

Exploratory Factor Analysis

> *Solving the number of factors problem is easy, I do it everyday before breakfast. But knowing the right solution is harder.*
>
> Henry F. Kaiser
> 1950s

Exploratory Factor Analysis is particularly relevant to Hypothesis 2a (Figure 4.3), which suggests that factors impacting health in open source software can be identified through publicly available data. As informed by the definition of health from Section 3.2.2, the factors at the project level are: engagement, interest, and robustness. There is no straightforward way to measure a subjective quality such as engagement, so the search looks for a latent factor that represents engagement based on collected metrics.[1] A similar fashion follows for interest and robustness subsequently in Chapter 7.

6.1 EFA for Developer Engagement

Exploratory factor analysis is a multivariate statistical method introduced in Section 4.3.2 used to identify latent constructs within a dataset. These constructs, or factors, signify data groupings that are either theorised or recognised but remain unobservable (Clark, 2018). The concept of developer engagement is explored presently via the six-stage approach of Hair Jr. et al. (2014) in Figure 4.5.

EFA is used here because the goal is to identify latent constructs for building a theory of OSS blockchain health (Fabrigar & Wegener, 2012). Additionally, EFA allows for correlation between latent variables which is to be expected in the confirmatory factor analysis stage of structural equation modelling (Chapter 8).

6.2 Stage 1: EFA Objectives

The first stage of the EFA is to identify the research problem which in this case is using open source data to search for underlying factors, or dimensions, that are not directly observable. The definition of health (Figure 3.6) identifies potential factors to be: engagement, interest, and robustness. The first round, presented in this chapter, identifies developer engagement and the subsequent round continues to find factors for interest and robustness (Chapter 7).

The research is taking an exploratory approach (not confirmatory), as part of the discovery process for a structural equation model (Chapter 8). The goal here is to achieve data summarisation rather than reduction as there are few indicator variables to assess. Data summarisation is concerned with finding structure within the set at a higher level than the individual variables. Data structure captures the interrelatedness of the variables allowing for a smaller set of factors overall.

1. The main result of this chapter is published in the *56th Hawaii International Conference on System Sciences* (Nijsse & Litchfield, 2023a).

As variables are being grouped, this study is an **R**-type factor analysis (Hair Jr. et al., 2014) used to identify structure by summarising the correlation matrix in the set of variables. **Q**-type factor analysis is less common and also analyses the correlation matrix to identify structure across respondents, or clusters within the data.[2] Within software engineering, the set of OSS projects is limited to blockchain-based software to reduce the influence from data clusters because adjacent fields (such as mobile development) may have different foundational constructs. This reinforces the choice of **R**-type analysis.

6.3 Stage 2: Factor Analysis Design

Designing a factor analysis involves variable selection and sample size assessment such that a suitable correlation matrix, **R**, can be assessed. For a mathematical description of SEM, see Appendix A.

6.3.1 Variable Selection

Selecting variables needs to accomplish two goals. First, there must be enough information to summarize in the form of a factor. Having a single variable be indicative of a factor does not help because the variable would de-facto be the factor. Having twenty possible indicators for a factor may not be guided by the researcher's expertise (or fit the exploratory criteria), and additionally makes the data collection impractical and expensive.

Best practices for factor derivation are to have three indicators per dimension (Watkins, 2018; Fabrigar & Wegener, 2012) and to better assist with structure identification for exploratory purposes at least five indicators per dimension (Hair Jr. et al., 2014, p.100). A single factor consisting of four variables is considered over-identified; three variables is just-identified; and two variables is under-identified. Factor η_1 in Figure 4.4 is considered just-identified, and η_2 is under-identified. It is best to avoid situations of under-identified factors.

Secondly, the data must be amenable to correlation matrix calculation. The nuance here lies in the distinction between metric data, that is, regular numerical values that exist on a scale, and non-metric data, such as feelings, colour, and gender. There are alternate methods available to handle non-metric data but are not required here as all indicators are metric and exist on a linear scale, for example, four months is twice as long as two months.

2. An example of a classic **Q**-type analysis would be to look for gender differences between respondents.

6.3.2 Sample Size

Sample size is a debated issue among statistics researchers, however, general sizes do emerge in the realm of EFA. With small sample sizes, there might not be enough information to properly estimate all of the parameters, leading to improper solutions such as negative variances. There is generally no upper limit as resource constraints of time and cost will prevent voluminous samples, however, the size scales based on the number of variables. In a meta analysis of 180 different populations conducted by Mundfrom, Shaw, and Ke (2005), given just-identified factors (3 indicator variables), the sample size of $n = 120$ is considered good and $n = 600$ is considered excellent.

Considering that SEM is the next stage of the research, the sample size should be adequate for both EFA and SEM. At the low end for SEM a sample of 150 can be useful (Wolf, Harrington, Clark, & Miller, 2013) but the size depends on the number of variables, the strength of indicator relationship, and factor overdetermination (Watkins, 2018).

A per-variable approach suggests a range from a minimum of 5 observations per variable to as many as 20 per variable (Hair Jr. et al., 2014). For example, when looking for engagement using 8 indicator variables times 20 observations is $n = 160$. This sets a lower bound for EFA. More observations are used here because of the higher requirements for SEM (Chapter 8), especially under non-normal data conditions. Additionally, Langer (2019) and Hair Jr. et al. (2014) suggests a sample size of $n = 200$ can correctly estimate the population when adhering to the design objectives. Finally, considering the potential for cross validation of the dataset, this doubles the requirement, and so a target data collection size of $n = 400$ is the goal.

The goal of this first round of EFA is to identify at least one factor beginning with at least three indicator variables, from a dataset of at least 400 observations (blockchain projects). The population in this case refers to the set of blockchain projects for which version control data is available, and is detailed next.

6.3.3 Data Collection

The factor analysis design stage answers how many, and what type of variables are to be included. The specific metrics needed to capture the characteristics of health are detailed in Chapter 3. Next comes the data collection stage. Goeminne and Mens (2013) provide a framework for extracting and analysing open source data that consists of: extraction from available data sources, collation into a database, data processing, and the output whether it be statistical analysis, visualisation, or reporting.

To find version control information for OSS blockchain projects, CoinMarketCap tracks the industry, beginning in 2012. They list over 20,000 tokens ranked by default by their

project's market capitalisation. Within each token project, CoinMarketCap lists the location of the project's source code repository. Using this information the top 600 blockchain projects are identified as of March, 2022 and collected through CoinMarketCap's API. Details retrieved include project `name`, `rank`, `website`, and `location of source code` if available. The data is collated into a Pandas dataframe (version 1.4.2) for Python (version 3.8.10) via JupyterLab (version 3.3.3).

Once projects are identified the source code history is needed to collect version control information. The platforms GitHub, GitLab, Bitbucket, Subversion, and SourceForge are public cloud-based open sourced repositories where people can independently work on coding projects or collaborate with others.[3] Predominantly, code hosting is handled by GitHub; and the study finds that 98.6% of the top 419 public repositories are hosted here. GitLab[4] and Bitbucket[5] host a minority of blockchain code, while SourceForge and Subversion host none.

Public blockchain project data can easily be accessed and analysed from GitHub. This information is available via the web interface, through the platform's API, or as raw archives from the GitHub Archive. All GitHub data from February 2011 is archived in `JSON` Lines format (`JSON` object on every line) and is available for public download from GHArchive. Every `JSON` object contains the metadata and data for one GitHub event. For example, when a repository is starred an event is emitted of `type:stargazer`. Similarly an event is created when a pull request is created, and the `JSON` contains all the details about who created it, when it is created, and the contents of the PR object.

GitHub currently has 17 event types[6], seven of which are relevant to this study:

- `ForkEvent`,
- `PushEvent`,
- `WatchEvent`,
- `PullRequestEvent`,
- `IssueCommentEvent`,
- `CommitCommentEvent`,
- `PullRequestReviewCommentEvent`.

A description of the event types follows in Section 6.3.5. All the events have metadata with `authorUsername` and `date` which can be used for further metrics.

GHArchive data is downloaded from February, 2011, up to 26 March, 2022, consisting of 2.36 TiB in total information. The compressed `JSON` is then inserted into a single-table ClickHouse database (Milovidov, 2020). ClickHouse is an open source column-oriented database

3. Section 3.1.2 details the history and workflow of using GitHub
4. `https://gitlab.com/`
5. `https://bitbucket.org/`
6. `https://docs.github.com/en/developers/webhooks-and-events/events/github-event-types`

management system designed for online analytical processing. This is ideal for large datasets that involve mostly read-only queries and batch updating.

The raw database contains 5.6 billion records—for all GitHub activity—is 430 GiB and is accessed with Structured Query Language (SQL) queries[7] through a command line or a Python module. This is running on a dedicated Linux Ubuntu (version 20.04.4) machine with ClickHouse's command line client and server (versions 22.3.3.44) installed.

6.3.4 Data Processing

In many cases the source code location is incorrectly reported and so all project source code locations are manually verified. Where the location points to an organisation on GitHub, the repository (repo) with the `reference`, or `core`, or `node` implementation is chosen. If this is not the case (perhaps it is not a blockchain), then the `contract` repository is chosen. To disambiguate between competing repos, the one with the most stars is chosen. Often the core repo is also the one with the most stars. When there are two implementations in different languages (for example, GO and Rust) highest stars takes preference. Forked libraries are not considered.

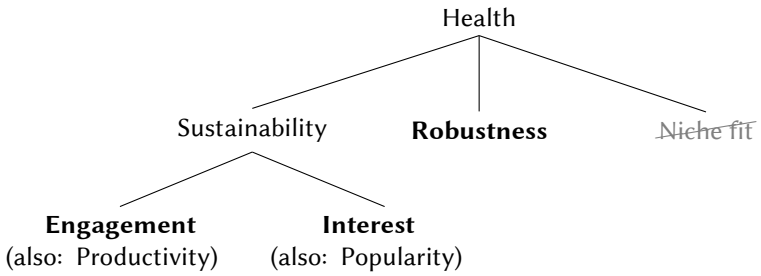


FIGURE 6.1: Modified software health definition focussing on the project level, excluding niche fit. The EFA searches for factors representing Engagement, Interest, and Robustness.

As an example of verification, the Synthetix ecosystem has six pinned repositories and is listed by CoinMarketCap as having code at `https://github.com/Synthetixio`, however, this is the organisation landing page and contains links to all repos. The main platform is hosted at `https://github.com/Synthetixio/synthetix`, which is manually verified.

Data cleaning involves excluding invalid projects by the criteria in Table 6.1. There is no missing or null data to handle as the absence of a metric is scored with a zero. For example, if there are no comments posted in the previous three months the query returns a zero. In Chapter 7, new metrics are introduced and techniques to handle the presence of missing data are discussed in Section 7.4.

7. The SQL queries are available in Appendix B

TABLE 6.1: Dataset cleaning and exclusion criteria beginning with the top 600 blockchain projects.

Total Projects	600
Exclusion Criteria	*Count*
No version control data The code cannot be accessed for analysis. 8 have a repo that's missing (404 error) indicating it has been deleted or moved; 78 are listed but private; 90 do not have a repo listed (and are likely private).	176
Alternate hosting service GitLab has four projects and Bitbucket has two. These are excluded because they are a small percentage of the whole (1.4%) and would require separate infrastructure to access the code bases.	6
Repository is listed twice Doubles are where the project points to the same code base for a related project; only the highest ranked project is included. KavaSwap (SWP) and KAVA Terra US dollar (UST) is double of LUNA Karura (KAR) is double of ACALA Shien network (SDN) is double of ASTAR Mirrored ProShares (VIX) is double of Mirror (mIAU) Steem dollars (SBD) is double of STEEM Theta fuel (TFUEL) is double of THETA	7
No contribution history There is no contribution history indicating the repo was created or the code was copied there and never updated; exclusion criteria is `Total Authors=0`. These are excluded as they are not representative of developer engagement in OSS.	26
Total after all exclusions (not mutually exclusive)	393

6.3.5 Data Metrics for Engagement

Recall the definition of health found in Chapter 3 and shown in Figure 3.6. The leaf nodes are engagement, interest, robustness, and niche fit. There is not enough justification to include the niche fit factor in the EFA as it is a broad ecosystem category inappropriate for the specific software repository level. Figure 6.1 shows the modified version whereby the leaf nodes are hypothesised factors to search for. The first round focusses on engagement.

A broad definition of engagement is "average hours per week worked on OSS projects" by developers (Poba-Nzaou & Uwizeyemungu, 2019, p.175), including during leisure time and work time (Schroer & Hertel, 2009). If they interact with GitHub during that work time, then any events registered by the version control software are seen as contributing to engagement.

GitHub has seventeen event types that get logged, and these can further be combined using metadata such as `username` and the `timestamp` to determine indicators of engagement.

Many engagement-type activities that developers and software communities participate in can map to GitHub events. The straightforward ones are traditional developer activities such as improving quality, responding to issues, fixing bugs, and implementing new features, and are captured by the `push`, `pull request`, and `comment` activities (Shaikh & Levina, 2019; Coelho, Valente, Silva, & Hora, 2018; Lee, Carver, & Bosu, 2017; G. Pinto, Steinmacher, & Gerosa, 2016). A number of studies use `author` information to derive metrics (Tamburri et al., 2019; Raja & Tretter, 2012; Shaikh & Levina, 2019); communication and discussion information through `comments` (Tamburri et al., 2019; Hata et al., 2022); `issues` to calculate response time and as a parallel to average mutual information (Raja & Tretter, 2012); and bookmarking of projects of interest through `forks` and `stars` (Schroer & Hertel, 2009). Tamburri et al. (2019) use `authors`, `pull request comments`, `commit comments`, and monthly active members, among seven total indicators to represent engagement.

Building on these studies, eight metrics are selected that meet the criteria of having a public repository hosted on GitHub and being potentially indicative of OSS engagement at a developer level. Responding to bugs is not directly measured here as it is a second-order result of baseline developer activity such as forking, PRs, and commenting. Response time is a metric for robustness; see Section 7.3.

Choosing a timeframe for certain metrics is important as projects grow and develop. Not all activity from the past is indicative of present performance, thus activity from twelve-months ago may not be relevant today. Using the most recent three months helps identify active projects and avoids the peril of dead projects (Kalliamvakou et al., 2016). Conversely, shorter timeframes such as one month can be too limited and fail to capture development milestones in progress (Z. Wang et al., 2020). From the previous three month's activity, a monthly average is computed.

Using the known location of the source code, a script is then written to query the database and calculate the following eight metrics. The main script is in Appendix B.2.

Stars A `WatchEvent`; the total count of stars received since creation. Any user with an account can star a project. This is similar to the like button or a bookmark. The highest number of stars in the dataset is for Bitcoin; there is no minimum requirement for inclusion, a project starts at zero. The data is collected using a SQL query[8] from the custom database.

Forks A `ForkEvent`; The total number of times a repository is forked since creation (see

8. See Appendix B for all the SQL queries.

Figure 3.2). The maximum number of forks in the dataset occurs for Bitcoin and there is no minimum requirement for inclusion of this metric, the floor is zero. Data is collected using a SQL query from the custom database.

Commits A `PushEvent`; the total number of commits to the codebase for the previous three months, calculated as a monthly average.

Pull Requests The count of `PullRequestEvents` for the previous three months, calculated as a monthly average.

Comments Comments are divided into three separate events: the sum of `IssueComment Event, CommitCommentEvent`, and `PullRequestReviewCommentEvent`; for the previous three months, calculated as a monthly average.

Authors The total number of unique authors by `username` that engaged with the repository at some time in the previous three months, calculated as a monthly average over the following event types: `PullRequestEvent, IssuesEvent, IssueCommentEvent, PullRequestReviewCommentEvent, and PushEvent`. The authors floor can be zero if no author interacted with the repository in the last three months.

Total Contributors The total number of unique authors that engaged with the repository since its creation. Includes the event types: `PullRequestEvent, IssuesEvent, IssueCommentEvent, PullRequestReviewCommentEvent, and PushEvent`. This is an aggregate metric providing a sum total proof of work that has gone into a project. The minimum is 1, a single contributor has to initialise the repository.

Days Inactive The number of days since the repository received an update, for example, 0.003 in Table 6.2 means the repo was updated 4.5 minutes before the query date. As this is a positive indicator, this metric is reverse-scored to be positively valenced, in other words, directionally consistent with the other indicators. The maximum value in the dataset is 1314.6 and means the repository went without update for 1314.6 days.

The dataset is available on GitHub[9] and contains complete data for 393 blockchain projects on which this chapter's EFA is based. The descriptive statistics for the dataset are shown in Table 6.2.

9. `https://github.com/millecodex/maui`

TABLE 6.2: Descriptive statistics for the cleaned dataset for eight indicator variables that may contribute to engagement in OSS blockchain health.

$n = 393$	Mean	Std. Dev	Minimum	Q1	Median	Q3	Maximum
PRs	14.1	37.2	0	0	2.0	13.7	524.3
comments	64.4	205.5	0	0	1.3	33.3	2440.7
authors	8.4	18.4	0	1	2.7	8.3	174.0
commits	92.2	205.4	0	0	8.7	81.7	1522.0
contributors (total)	135.8	543.1	1	8	24	76	7465
inactive (days)	50.8	156.9	0.003	0.6	2.4	19.9	1314.6
forks	528.0	3398.5	0	12	59	204	59 013
stars	772.8	4350.1	0	22	101	391	72 112

6.4 Stage 3: EFA Assumptions

The key assumptions that are made at this point involve conceptual decisions and statistical assumptions. Conceptually, there is an expectation of structure within the data, and this is supported by the literature review. Because factor analysis always yields factors, sound judgement in the selection of metrics is important. Additionally, data homogeneity is ensured by selecting a subset industry of the GitHub ecosystem. The selection must be broad enough to meet the required sample size, yet narrow enough for resource constraints. Blockchain-based projects including blockchains, cryptocurrencies, decentralised exchanges, tools, layers, and bridges, are selected. Other industries such as web development, libraries, and mobile development are excluded to limit the domain and potential correlation corruptions to the factor structure that subsamples could influence.

The statistical assumptions that are made before conducting the factor analysis include the distribution of the data and assessment of the correlation matrix.

Data Distribution

Data distribution is important when considering what methods and statistical tests are to be applied to the dataset and how this can affect downstream analysis. EFA does not require data to have a particular distribution, however, the chi-square test of goodness-of-fit, used to test the adequacy of a factor model, is sensitive to departures from normality (Hair Jr. et al., 2014).

In cases of substantial non-normality, there are strategies that can be adopted such as robust estimation methods that are less sensitive to departures from normality, or data transformation methods. This is addressed in Section 6.6.1. The data in Table 6.2 are highly non-normal, the Cullen and Frey plots in Figure 6.2 show most variables have a kurtosis and skew

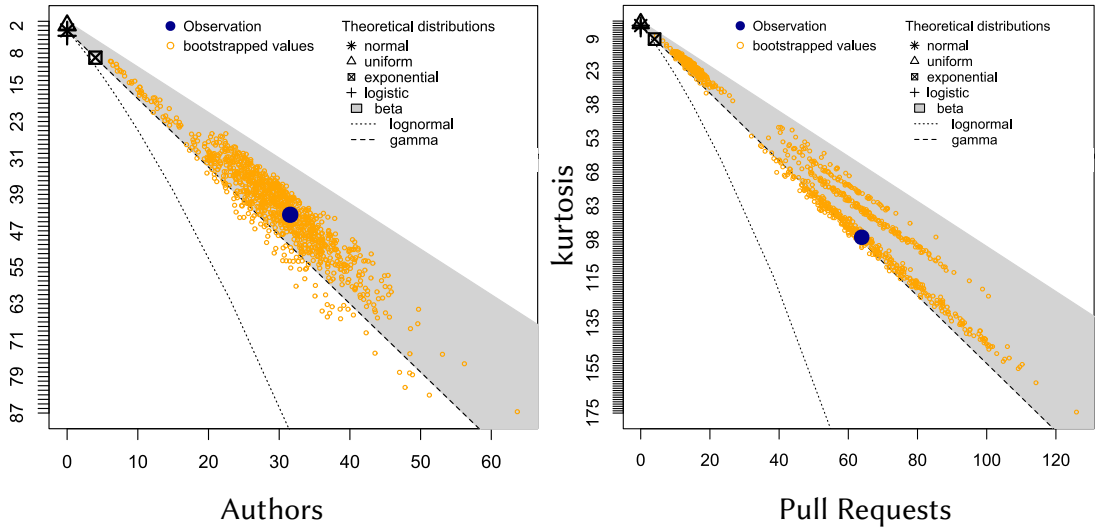


FIGURE 6.2: Cullen and Frey plots for authors and pull requests. Square of skewness is on the x-axis and kurtosis on the y-axis. The blue dots indicate where the observed data sit, the yellow dots are bootstrapped simulations ($n = 1000$) employing the same descriptive statistics. Both are consistent with a beta distribution; PRs is close to a gamma distribution. Generated using `descdisc` from the `fitdistrplus` package in **R**.

relationship consistent with data in the beta distribution range. Standard distribution plots for all eight indicator variables are in Figure 6.3, and the logarithmically scaled distributions are in Figure 6.4.

All variables should be directionally consistent to reduce the effect of skewness influencing the factors (Norman & Streiner, 2008). The days inactive variable is reverse-scored so that a lower value has a negative effect and a higher value indicates a positive effect.

Correlation Matrix Suitability

Tests to justify the application of EFA to the correlation matrix (Figure 6.5) include Bartlett's test of sphericity (Bartlett, 1950) and the Kaiser-Meyer-Olkin (KMO) test (Kaiser, 1974), also called the measure of sampling adequacy. Bartlett's test provides a single value for correlation of the data within the matrix, the result of $\chi^2(393) = 4400$, at a significance of $p < 0.001$ indicating the data is suitable for EFA (anything $p < 0.05$). Bartlett's test is sensitive to sample size, and so the KMO test is also used. The KMO test is a score between 0 and 1, with 1 representing perfect correlation between the variables. Here, the overall KMO is 0.77 and the general rule is that each indicator variable should be above a 0.5 threshold. A value lower than this indicates the data is too random for EFA. Table 6.3 shows the factor adequacy values.

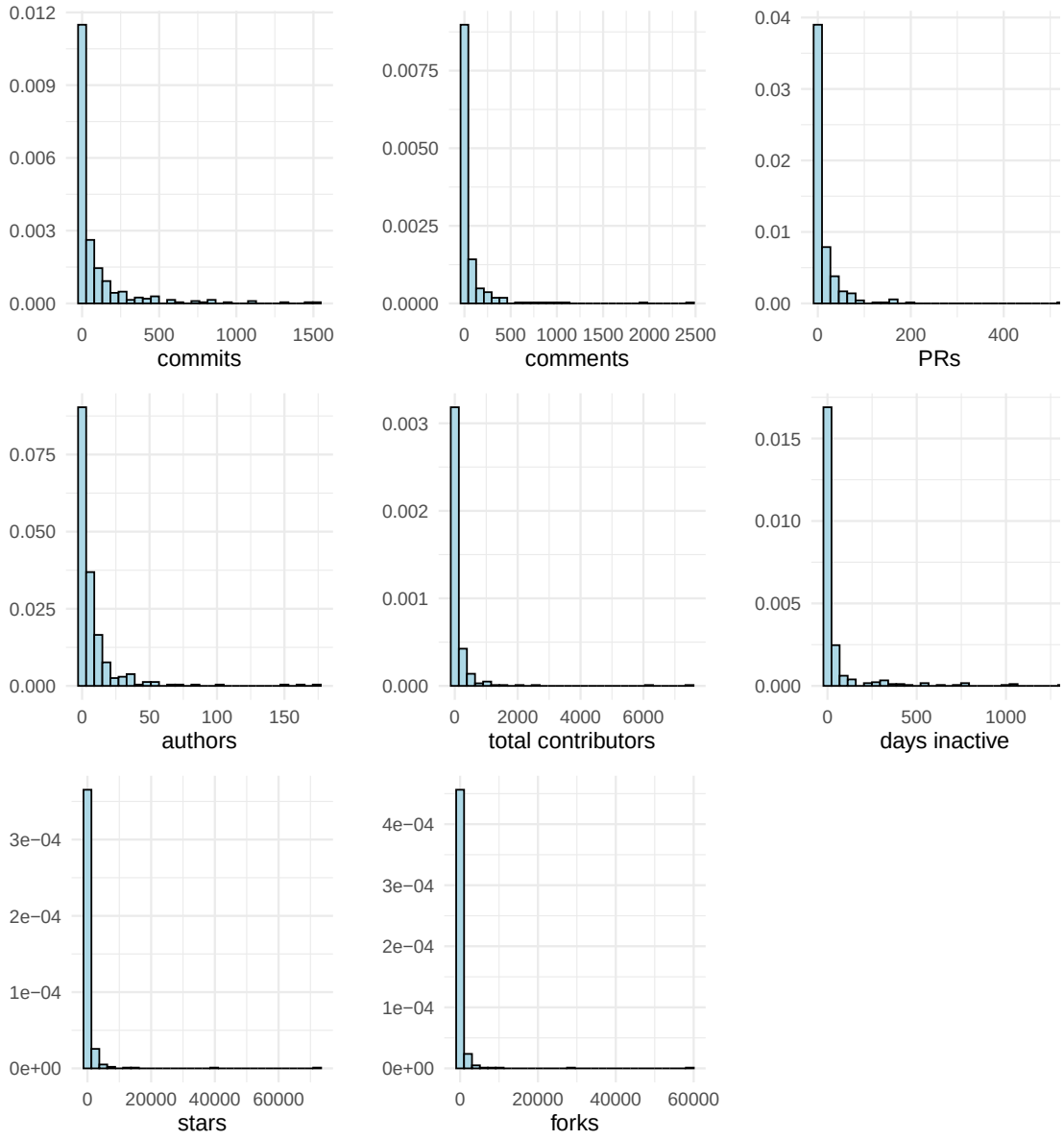


FIGURE 6.3: Histogram distributions of metrics from the dataset. The $y-$axis represents the density, which is the proportion of observations within each bin. All variables are highly skewed and non-normal.

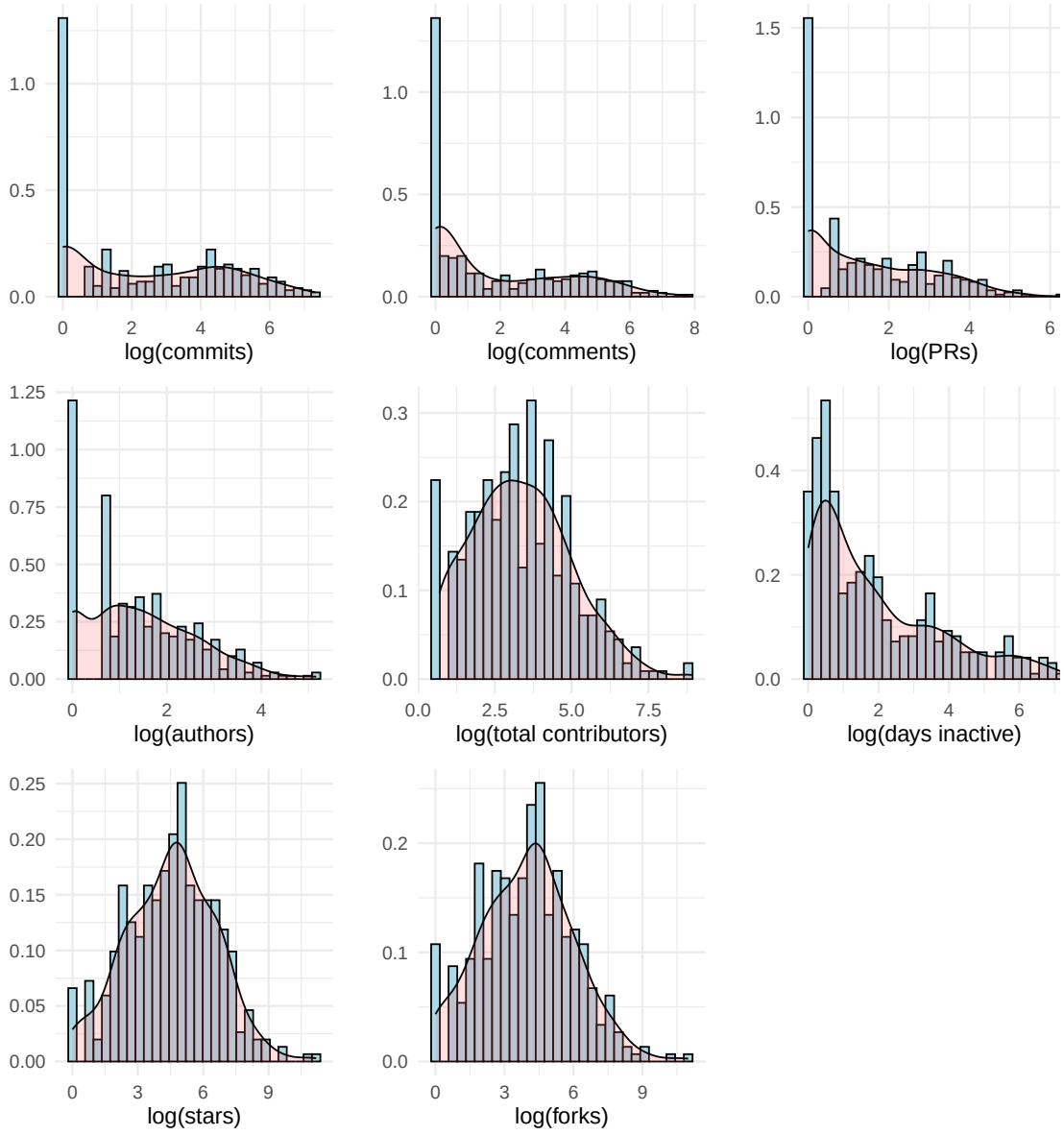


FIGURE 6.4: Logarithmically scaling the data allows for closer inspection of the distribution. The $y-$axis represents a probability density function for a continuous random variable. The density plot is a smoothed version of a histogram estimated from the data.

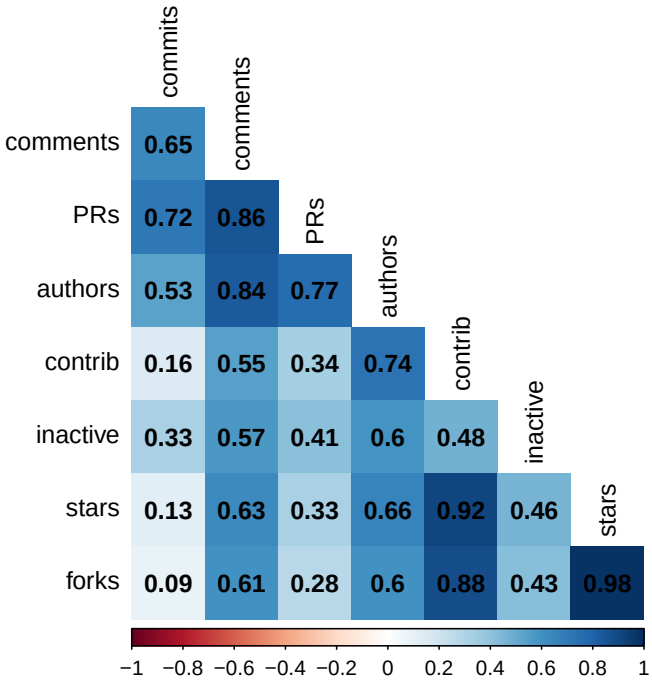


FIGURE 6.5: The Pearson correlation matrix for the dataset shows many strong correlations as input data for the EFA.

TABLE 6.3: Kaiser-Meyer-Olkin test scores for individual variables in the dataset all show values above the 0.5 cut-off.

Factor	Stars	Forks	Contrib.	Authors	Commits	Comments	PRs	Inactive
Value	0.77	0.75	0.72	0.82	0.85	0.71	0.79	0.88

The dataset has met the requirements, both conceptual and statistical to proceed to the factor derivation stage.

6.5 Stage 4: Factor Derivation

Deriving factors from the dataset involves selecting the method of factor derivation, either component analysis or common factor analysis, and then determining the number of factors to extract.

As stated in Section 6.2 (Stage 1), the concern is with identifying structure and so common factor analysis is used. The difference between component analysis and common factor analysis lies in the type of error variance that is present. Common factor analysis extracts common variance (the amount of variance that is shared, or correlated, with all other vari-

ables) and is represented by the term communality (Hair Jr. et al., 2014). Component analysis extracts total variance which includes communality but also specific variance, that which is not attributable to other variables, and error variance, due to unreliability in the data, more common in surveys.

How many factors to extract?

An a priori criterion of a single factor for extraction—developer engagement—is the basis of the first round of factor derivation. Secondary factors are not excluded, rather are investigated in the second round of the EFA (Chapter 7). Multiple iterations of factor analyses are undertaken as required, such as for the removal of indicator variables.

Scree plots are examined to see the number of proposed factors, keeping in mind that they are known to be inaccurate for determining the number of factors to retain (Finch, 2020b). A better method is to use a parallel analysis which employs random data with the same properties (mean, standard deviation) as comparators. Should there be strong deviation of the actual from the simulated data, then a statement can be made to support the number of factors. The latent-root criterion[10] sets a cut-off at an eigenvalue of 1, keeping factors that are >1.

Both a standard scree and parallel analysis in Figure 6.6 indicate strong preference for a single factor and weak preference for a second factor. The parallel analysis has significant deviation from the random data at eigenvalue $\lambda_1 = 4.63$ for the first factor and $\lambda_2 = 1.23$ for the second compared to simulated values of 0.48 and 0.16 respectively. Factor three has nearly identical eigenvalues as computed from the data and compared to simulated values meaning that if a third factor is chosen, this exhibits no more structure than would be expected from random (non-correlated) data.

In summary there is strong support for a single factor, to be compared against both two- and three-factor models to gauge efficacy. Increasing the number of extracted factors can provide context and clarity in knowing when to stop.

6.6 Stage 5: Factor Interpretation

Factor interpretation is the researcher's assessment of the factor matrix including judgement on rotation methods, statistical tests, and iterative refinements. The iterative portion is subjective in nature and relies on the solid theoretical basis in Chapters 2 and 3 and factor analysis design in Section 6.3.

10. At this step only a single factor is considered.

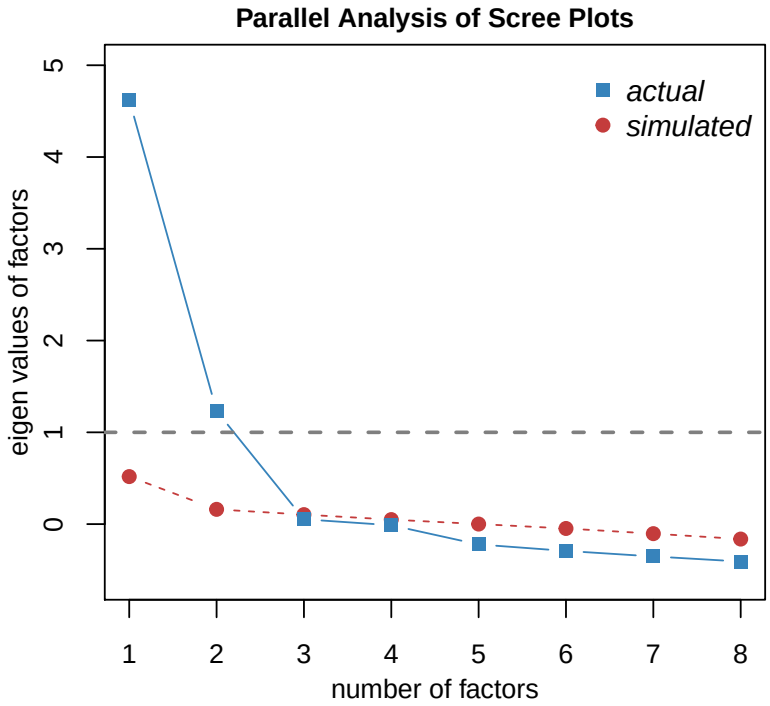


FIGURE 6.6: Parallel analysis results indicating strong support for a single factor ($\lambda_1 = 4.63$) and mild support for a second factor ($\lambda_2 = 1.23$). The dashed line shows the latent-root criterion cut-off of 1; factors with values above 1 have variance that is associated with the factor while values below 1 could be attributed to random correlation. The scree plot criterion places the elbow at three factors, which, in this instance would retain two factors.

6.6.1 Factor Matrix Estimation

Factor analysis is carried out with the `psych` package (version 2.2.3) in **R**.[11] Differences in output factor loading occur due to different optimisation algorithms used by the packages; these are estimation methods and do not have analytical solutions. `lavaan`'s EFA is in beta and is useful to understand the syntax to forward apply to SEM (Chapter 8). `psych` has more features, documentation, and support, and so is chosen for the EFA. The computational method used to estimate the factors is maximum-likelihood (ML) known to perform well when the factor–variable relationships are strong. The principle-axis method is also used for comparison purposes as it is ideal for non-normality and small sample sizes (Watkins, 2018). In cases of substantial non-normality a robust ML estimation method is less sensitive to departures from normality and provides more reliable estimates of factor loadings and standard errors.

11. The EFA is also carried out using the `lavaan`, and `epmr` packages with similar results. Only `psych` results are reported presently, in Chapter 8 `lavaan` results are used for SEM.

6.6.2 Factor Rotation

The unrotated factor matrix can limit the interpretation of factors because variables often load heavily on more than one factor, making it difficult to discern the distinct dimensions of underlying constructs. Rotation helps to redistribute the loadings of the variables on the factors with the goal of achieving a simple, clearer dimensional structure.

Factor rotation is done with the `GPArotation` package (version 2022.4-1). The Varimax factor rotation method maximizes the variances of the loadings within the factors. This can help with structure for two or more factors. Conversely the Quartimax method is a better choice for detecting an overall factor as it maximizes the sum of squared (SS) loadings so that each item loads most strongly onto a single factor. For this reason, Quartimax rotation is used with resulting stronger loadings applied to the first factor.

Both Quartimax and Varimax are orthogonal rotation methods and therefore the amount of variance explained by each factor is reported independently, compared with oblique methods, which allow factors to correlate and share variance between the factors.

6.6.3 Factor Interpretation & Respecification

The iterative process describes the model selection, rotation method selection, and factor number comparison process. This leads to elimination of indicator variables and the analysis is completed again.

Factor Estimation Method

Table 6.4 compares the maximum likelihood and principle axis methods setting the rotation method to none. Without rotating the factor solution first the maximum likelihood is compared to the principle axis method. ML has a better Bayesian Information Criterion (BIC) model fit and lower root mean square error of approximation (RMSEA) for the ML version, additionally the conceptual distribution of indicators in the ML model is more suggestive of structure. This provides support to use maximum likelihood going forward. Before moving to assess factor matrix rotation methods, the EFA output is explained.

EFA Output

The **communalities** (h^2 in Table 6.4) or common variance refers to the proportion of variance in the observed variables that is explained by the corresponding latent factor. It represents the proportion of the variance in the observed variable that can be attributed to the factor, after accounting for measurement error. For instance, forks has an h^2 of 0.968 for the ML model,

TABLE 6.4: Estimation method comparing maximum likelihood to the principle axis estimation methods with no factor matrix rotation. Two factors are extracted. Boldface indicates significant factor loadings showing a difference in factor structure between the two methods. ML has a better BIC model fit and lower RMSEA, additionally the conceptual distribution of indicators in the ML model is more suggestive of structure. This provides support to use maximum likelihood going forward.

Estimator	Maximum Likelihood			Principle Axis		
Variable	ML_1	ML_2	h^2	PA_1	PA_2	h^2
stars	**0.997**	−0.046	0.995	**0.860**	−0.504	0.993
forks	**0.980**	−0.089	0.968	**0.813**	−0.518	0.929
contributors (total)	**0.917**	−0.010	0.841	**0.824**	−0.426	0.861
authors	**0.691**	0.571	0.804	**0.903**	0.169	0.844
inactive (days)	**0.480**	0.321	0.333	**0.593**	0.058	0.356
PRs	0.371	**0.884**	0.920	**0.723**	0.598	0.880
comments	0.667	**0.685**	0.915	**0.901**	0.324	0.917
commits	0.163	**0.757**	0.599	0.495	**0.596**	0.600
SS loadings	4.111	2.264		4.826	1.553	
Cumulative Variance	0.514	0.797		0.603	0.797	
Proportion Explained	0.645	0.355		0.757	0.243	

indicating that 96.8% of the variance in forks data can be explained by the first latent factor, ML_1.

SS loadings in the output of the factor analysis refers to the sum of the squared (SS) factor loadings for each latent factor, which represents the amount of variance in the observed variables that is accounted for by each factor. Specifically, the SS loadings represent the sum of the squared factor loadings for each variable on each factor, weighted by the variance of each variable. In the output, the SS loadings are presented separately for each latent factor, and they indicate the amount of variance accounted for by that individual factor.

For example, in the output above, the SS loadings for the ML model in Table 6.4 are 4.111 and 2.264. This indicates that Factor ML_1 accounts for 4.111 units of variance in the observed variables, and Factor ML_2 accounts for 2.264 additional units of variance in the observed variables. These values can be used to calculate the proportion of variance explained by each factor, which is presented in the Proportion Explained output.

Cumulative variance refers to the total amount of variance in the observed variables that can be explained by the factors up to that point. In this example, the cumulative variance for ML_1 is 0.514, and the cumulative variance for ML_2 is 0.797 so the proportional variance for ML_2 is 0.797−0.514=0.283.

Proportion explained refers to the proportion of the total variance in the observed variables that is explained by each factor. It is calculated by dividing the SS loading for the

TABLE 6.5: Two factors are extracted using ML to compare QUARTIMAX and VARIMAX rotation methods. QUARTIMAX is a better choice for detecting an overall factor as it maximizes the SS loadings (3.501 > 3.406) so that each item loads most strongly onto a single factor. High cross-loadings are seen for comments, authors, and days inactive in both models leaving these as inconclusive indicators.
† the loading on inactive in ML_{Q2} of 0.413 is much lower than the remaining indicators and is subsequently removed.

Rotation Method	VARIMAX			QUARTIMAX		
Variable	ML_{V1}	ML_{V2}	h^2	ML_{Q1}	ML_{Q2}	h^2
stars	**0.979**	0.190	0.995	**0.985**	0.161	0.995
forks	**0.973**	0.144	0.968	**0.977**	0.115	0.968
contributors (total)	**0.893**	0.207	0.841	**0.899**	0.180	0.841
PRs	0.152	**0.947**	0.920	0.180	**0.942**	0.920
comments	0.487	**0.823**	0.915	0.511	**0.808**	0.915
commits	−0.020	**0.774**	0.599	0.003	**0.774**	0.599
authors	0.537	**0.718**	0.804	0.558	**0.702**	0.804
inactive (days)	0.391	**0.425**	0.333	0.403	**0.413**†	0.333
SS loadings	3.406	2.970		3.501	2.875	
Cumulative Variance	0.426	0.797		0.438	0.797	
Proportion Explained	0.534	0.466		0.549	0.451	

factor by the total variance of all the variables. In this example, the proportion explained for ML_1 is 0.645=4.111/(4.111+2.264), and the proportion explained for ML_2 is 0.355 (that is, 1.000−0.645).

Factor Rotation Method

Next, Table 6.5 uses ML to compare rotation between QUARTIMAX and VARIMAX methods. Model fit statistics do not differ when varying rotation methods, only the factor loadings as rotation is designed to emphasise structure. QUARTIMAX is a better choice for detecting an overall factor as it maximizes the SS loadings so that each item loads most strongly onto a single factor. Stars and forks are so highly correlated that this produces a strong factor structure and so QUARTIMAX puts this as the primary factor. The secondary factor, ML_{Q2}, is representative of the construct the method is looking for. The *inactive* indicator loading on ML_{Q2} of 0.413 is much lower than the remaining indicators and is removed during factor respecification.

TABLE 6.6: Maximum Likelihood with QUARTIMAX rotation comparing the number of factors extracted: one, two factors, and three factors. The indicators are ordered the same as the QUARTIMAX two-factor model in Table 6.5. h^2, the proportion of variance explained by the factor is left out. The addition of a third factor leaves no leading indicator by loading (absence of boldface) and accounts for only 3.4% of the total variance. Note that ML_{Q1} and ML_{Q2} have the same factor structure as in Table 6.5, shown here for easy comparison.

Factors Extracted	1	2		3		
Variable	ML_{11}	ML_{Q1}	ML_{Q2}	ML_{31}	ML_{32}	ML_{33}
stars	**0.998**	**0.985**	0.161	**0.972**	0.177	−0.085
forks	**0.982**	**0.977**	0.115	**0.971**	0.128	−0.176
contributors (total)	**0.917**	**0.899**	0.180	**0.934**	0.186	0.283
PRs	0.333	0.180	**0.942**	0.173	**0.915**	0.025
comments	**0.636**	0.511	**0.808**	0.478	**0.852**	−0.204
commits	0.130	0.003	**0.774**	0.000	**0.770**	0.033
authors	**0.666**	0.558	**0.702**	0.573	**0.718**	0.245
inactive (days)	**0.465**	0.403	**0.413**	0.397	**0.461**	0.068
SS loadings	3.993	3.501	2.875	3.504	2.964	0.226
Cumulative Variance	0.499	0.438	0.797	0.438	0.808	0.837
Proportion Explained	1.000	0.549	0.451	0.523	0.443	0.034

Number of Factors Extracted

In the subsequent analysis, ML estimation is utilised in conjunction with QUARTIMAX rotation to evaluate the optimal number of factors to extract, as delineated in Table 6.6. Specifically, the comparison encompasses one, two, and three factors.

Table 6.6 shows that the addition of a third factor leaves no leading indicator by loading and accounts for only 3.4% of the total variance. Thus, extracting a third factor is erroneous and does not provide new information to include in the model. The factor structure is retained between two and three factors (ML_{Q1} and ML_{31}), but gets scrambled for a single factor extraction due to the outsized influence of stars and forks.

Model Respecification

ML_{Q1} and ML_{Q2} from Tables 6.5 and 6.6 thus far represent the best candidates for representing underlying structure within the dataset.

This is shown as MODEL A in Table 6.7 and suggests a primary factor, A_1, composed of: stars, forks, and total contributors with a secondary factor A_2 being: PRs, comments, authors, commits, and days inactive. Levels of significance are determined with loadings > 0.4; all loadings are shown for completeness. The high communality, h^2, (and therefore low uniqueness)

TABLE 6.7: Two models side by side. Factors are reordered according to the new MODEL B showing the primary factor consisting of PRs, comments, authors, and commits. B_1 contributes to the main influence of the model (SS loading = 3.10) with 73% of the variance accounted for. B_2 has only a single indicator, with a modest SS loading = 1.13. Days inactive in MODEL A still exhibits high and inconclusive cross-loading which is reduced to insignificant in MODEL B.

	MODEL A			MODEL B		
Variable	A_1	A_2	h^2	B_1	B_2	h^2
PRs	0.180	**0.942**	0.920	**0.96**	−0.03	0.078
comments	0.511	**0.808**	0.915	**0.90**	0.22	0.140
authors	0.558	**0.702**	0.804	**0.81**	0.47	0.121
commits	0.003	**0.774**	0.599	**0.74**	−0.14	0.428
contributors (total)	**0.899**	0.180	0.841	0.38	**0.92**	0.014
inactive (days)	0.403	**0.413**	0.333	−0.13	−0.02	0.984
stars	**0.985**	0.161	0.995			
forks	**0.977**	0.115	0.968			
SS loadings	3.501	2.875		3.10	1.13	
Cumulative Variance	0.438	0.797		0.52	0.71	
Proportion Explained	0.549	0.451		0.73	0.27	

of the variance attributed to stars (0.995) and forks (0.968) suggests these indicators could measure the same thing and therefore be removed, or be indicative of a new factor. The SS loadings for each factor indicates a stronger relationship for A_1 (3.50) compared to A_2 (2.88). This is the motivation for the model respecification.

The model is revised to exclude stars and forks with the intent of reserving these indicators for an additional factor. Figure 6.7 shows the process of beginning with MODEL A and refining to arrive at MODEL B. Table 6.7 shows the revised factor analysis that retains the indicator variables from A_2, now as the primary factor (B_1) with a much higher SS loading than the secondary factor (B_1= 3.10 compared with B_2= 1.13). There is moderate cross loading on authors, as it has near-significant value of 0.47 loading on B_2. As the loading on B_1 is more significant, and B_2 is of secondary interest at this stage, this is acceptable. The cumulative variance explained by the two-factor model is 71% with B_1 responsible for 73% of that.

High cross-loading is present on days inactive for both MODEL A and MODEL B. In MODEL A it loads at 0.403 and 0.413 only showing a mild preference to belong to factor A_2. In MODEL B days inactive is slightly negative and close to zero. A zero loading indicates no affinity for the factor and is strong evidence that days inactive does not belong with these groupings.

Both MODEL A and B indicate total contributors should be a second factor and this is addressed in Chapter 7.

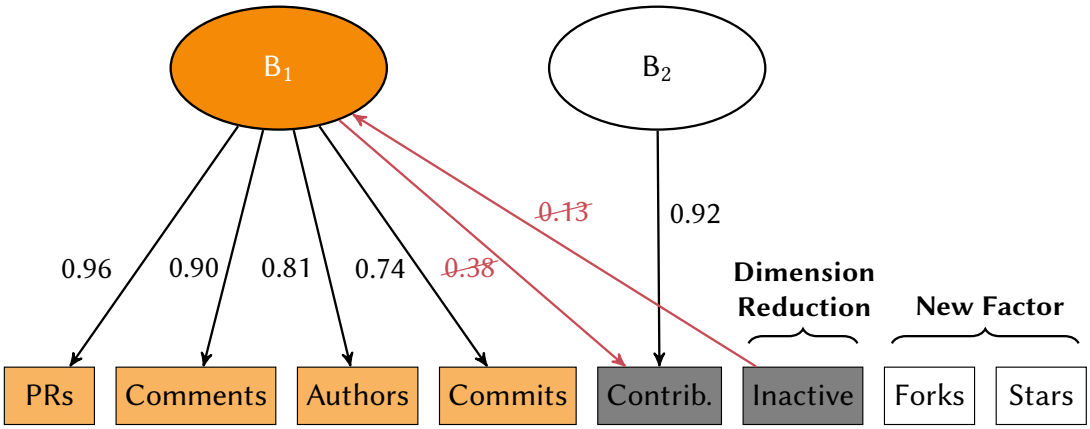


FIGURE 6.7: Model respecification. The indicator loadings are shown for MODEL B. Total contributors does not have a high enough loading and is excluded. Days inactive loads high on B_2 and is not included with B_1, and is removed for dimension reduction. Forks and stars are highly correlated from MODEL A and are removed to be included in a new factor (Chapter 7).

6.6.4 Fit Statistics

Individual fit statistics are not hard rules, nor provide enough evidence to be used in isolation and therefore it is recommended to use a suite of tests for comparison (Finch, 2020b). The fit statistics for the models are summarised in Table 6.8.

As this EFA is part of a discovery process for CFA and SEM, the CFA thresholds can be applied. Chi-square for MODEL A is $\chi^2 = 493.4$, $p < 0.001$ which indicates a poor fit and suggests the model could be improved. In contrast, the chi-square test is an indicator of badness-of-fit, and thus, a p-value greater than 0.05 is preferred, as seen in MODEL B ($\chi^2 = 7.289$, $p < 0.12$).[12] It's noteworthy that large sample sizes tend to produce small p-values for chi-square tests (Alavi et al., 2020); hence, while they are considered, they are not solely relied upon for determining model fit.

The Tucker-Lewis Index (TLI) compares the resultant model with a baseline where there is no factor structure. MODEL B is significantly better and meets the > 0.9 target (Finch, 2020b), and > 0.95 target for large sample sizes ($n > 250$) (Hair Jr. et al., 2014).

The RMSEA < 0.05 is a common level however this should not be a universal cut-off (F. Chen, Curran, Bollen, Kirby, & Paxton, 2008). The RMSEA value is revised upwards to < 0.07 when applied more to structural equation models, further recommending that model fit is better represented by a TLI or Comparative Fit Index (CFI) measure than a statistical fit test.

SRMR is the square root of the sum of squared correlation residuals for the variables.

12. $p = 0.01$ indicates there's a 1% probability that the observed data would differ from what the model predicts if the model is a perfect fit. Generally, a higher p-value suggests a better model fit.

TABLE 6.8: Fit statistics for the two models showing better overall fit for MODEL B. *** is $p < 0.001$ (insignificant), † is $p > 0.05$ (significant).

Fit Statistic	MODEL A	MODEL B
χ^2	493.4^{***}	$7.289^{\dagger}$
TLI	0.763	0.993
RMSEA	0.307	0.046
SRMR	0.03	0.01
BIC	415.74	−16.61

Also a badness-of-fit measure, SRMR > 0.1, indicates a good model fit, however, this is biased upward so can be ignored when only considering a few indicators as in this study (Finch, 2020b).

The BIC is a comparative indicator where lower values are better. MODEL B's BIC is less than MODEL A, supporting the other fit statistics, and providing evidence that the exclusion of stars and forks produces a better model.

6.7 Stage 6: Factor Validation

Model validation is by two mechanisms: CFA applied to the measurement model, and cross-validation by separation of the dataset into a training and testing segment. CFA is undertaken in Chapter 8, and thus cross-validation is relevant here.

Cross validation is necessary to avoid the situation where the model ends up being overfit to the data, affecting generalisability. Resource constraints in the data collection process, both on time, and on the number of available projects limit collecting an entire new dataset and so the original is split into two groups. Random allocation is performed using the `caTools` package (version 1.18.0) with 50% split to produce two groups, one to build the model, and one to test the model.[13]

Cross validation results are in Table 6.9. The insignificant loadings are not shown for readability and to highlight that the same factor structure is present across the models. The training and testing models are not as well fit as MODEL B based on χ^2, TLI, and RMSEA, however are acceptably close considering the sample size handicap.

Before concluding on the EFA, the assessment of influential observations is noted.

13. The original study is expanded to collect more data to get close to this minimum sample size threshold of 200 and is seen iteratively in Figure 4.5.

Table 6.9: Cross-validation for Model B showing equivalent factor structure for the testing model as compared to the base EFA. Insignificant loadings are not shown.

	Model B		Training		Testing	
Factor	B_1	B_2	Tr_1	Tr_2	Te_1	Te_2
commits	0.74		0.91		0.70	
comments	0.90		0.94		0.99	
PRs	0.96		0.97		0.96	
authors	0.81		0.84		0.89	
contributors (total)		0.92		0.41		0.87
inactive (days)						
n	393		195		196	
χ^2	7.289		9.77		25.74	
TLI	0.993		0.98		0.92	
RMSEA	0.046		0.086		0.166	

Assessment of Outliers

The data is examined for influential observations (outliers) and although there are extreme values, these observations represent actual blockchain projects and are included in the spirit of producing a representative model. Exclusion of potential outliers is tricky, and without good cause, such as impossible values or missing data, it is best to include them (Aguinis, Gottfredson, & Joo, 2013). Further, the EFA analysis is completed with and without some of the possible influential observations according to the Mahalanobis distance to no marked difference in results (Fidell & Tabachnick, 2003). This is good evidence to include all data. Figure 6.8 shows an example of the influential observations in the dataset.

Excluding data would have to be justified based on a data collection error, or the results being influenced. Mahalanobis D^2 is based on the assumption of multivariate normality, which this dataset does not meet, however Figure 6.8 can still provide visual cues within the data that something may need to be investigated. It is possible to have data with no actual outliers, but the distribution of D^2 values could still be skewed or have heavy tails because the data themselves are not normally distributed.

Thus, considering the effects from influential observation, and based on the identical structure of the testing and training models with Model B, there is confidence in the robust nature of the factor structure within the dataset.

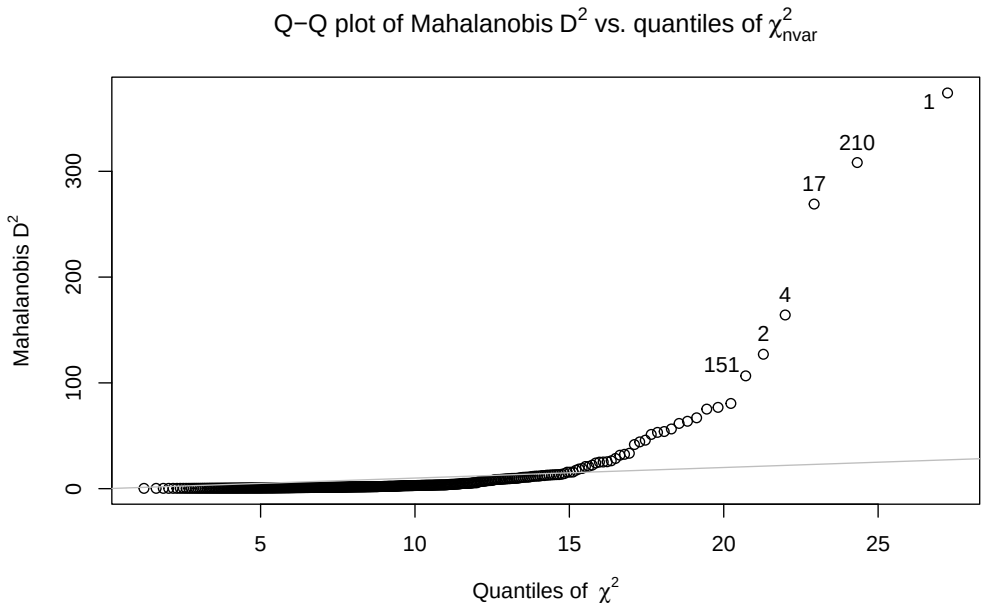


FIGURE 6.8: Influential observations in the dataset are present, but indicative of OSS blockchain projects, and thus included in the analysis. Here, projects are identified by number 1::Bitcoin, 210::Solana, 17::Ethereum, 4::Dogecoin, 2::Litecoin, 151::Cosmos.

6.8 Analysis: Model B → Engagement

Now that there is sound statistical reasoning for MODEL B presented in Table 6.7, the logical significance of the factor structure is assessed, justifying the respecifications along the way. Does it make sense? Can it be named? The highest loaded indicators often directly endow their etymology to the factor. Here, the researcher must be able to adequately name the factors, which is subject to interpretation and criticism (Hair Jr. et al., 2014).

Factor A_1 fits naturally as a community indicator of Popularity, or Community Interest, and is distinguished in the model from the more traditional developer activities. **Developer Engagement** remains the best naming option to capture A_2's latent construct as these events are more likely to be part of a developer's contribution than casual a community member's.

It is for this reason a two-factor analysis is used to be able to judge where to draw the line given the a priori condition of developer engagement. Does forking count as engagement or could it be better represented by something else? Both MODELS A and B conclude the same indicator variables for the dimension with second factor components being responsible for the statistical fit differences.

The high correlation of stars and forks is known in the broader software environment when assessing GitHub's overall most popular projects (Abdulhassan Alshomali, 2018; Borges,

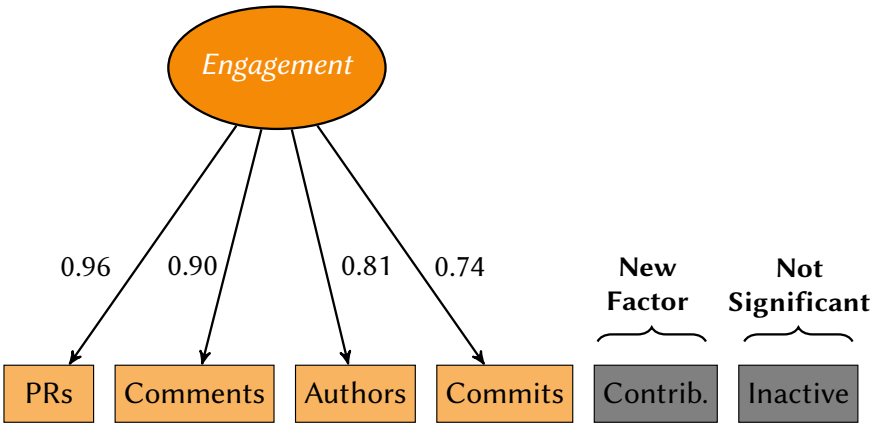


FIGURE 6.9: Factor analysis diagram updated from Figure 6.7; B_1 is named Developer Engagement while B_2 leads to a subsequent factor for total contributors, and days inactive remains insignificant.

Hora, & Valente, 2017). Figure 6.5 shows the correlation at 0.98 and this confirms the stars/-forks relationship for blockchain projects. Starring and forking are not active developer activities, rather are passive; many people browsing by a project may star it to bookmark the project, or fork it to explore the code.[14] Both activities are quick and easy to do, starring requires a single click and forking needs two clicks. They are much less intensive than commenting on an issue or submitting a pull request and can thus be categorised as a general population GitHub activity, whereas authoring updates is a classic developer activity.

Thus stars and forks are removed from the analysis (see Figure 6.7) leading to MODEL B diagrammed in Figure 6.9. The factor analysis is also done with the inclusion of forks (more or a developer-based activity) and exclusion of stars (more social-media-based activity), and vice versa. This does not affect the factor structure; stars and forks stand on their own as an independent factor.

Both MODEL A and B indicate total contributors should be a second factor and this is handled next in Chapter 7. MODEL A suggests this dimension includes stars and forks. A potential candidate name is *Public Interest* whereby if a project becomes interesting or popular, developers bookmark it (star or fork) and author a comment (which is counted in total contributors). However, the strength of the stars-forks bond may exclude contributor numbers all together, leaving an under-determined dimension.

The number of days since a repository is last updated (days inactive) does not influence either factor. In both models this indicator does not have a preference for either factor, and in

14. An operational note regarding the stargazer event, this is a unidirectional metric as there is no event for removing a star. Thus the reported number of stars by querying the database may be slightly larger than what appears on the website. In practice, few people remove a star. Large sample sizes mitigate these issues as the model is interested in capturing trends across many projects.

Model B its loadings are −0.13 and −0.02, both insignificant. As this is a time-based metric it may apply more accurately to time-critical dimensions such as time to resolve issues or merge pull-requests.

The strongest loading factor in Model A, Model B, and the cross-validation training model is PRs. Although not singularly dominant, the PR (in addition to issues tracking via comments) is seen as the primary way to update a codebase (Gousios et al., 2016). Given the ease with which GitHub facilitates the pull-based model (Gousios, Pinzger, & Deursen, 2014), it is justified to conclude that the latent construct named **Developer Engagement** is led by pull-request and commenting metrics.

To summarise, the statement of influence from Section 6.1, $\eta \rightarrow x$, can now be written as: developer engagement is positively related to pull requests and comment activity. Authors and commits also present positive influences, all on a recent three month timeline.

6.9 Conclusion

This work uses exploratory factor analysis to identify dimensions that represent engagement in a community of open source blockchain developers. Findings indicate that developer engagement can be represented by authors, commits, comments, and pull requests based on a monthly average of the previous three months. Pull requests is the single strongest influence indicator. Stars, forks, and total contributors are strongly loaded on a second factor hypothesised to be general interest and is investigated next. Cross validation of the dataset is carried out and shows the same factor structure emerges for developer engagement in a model with similar fit characteristics to the original.

This chapter focuses on finding a latent construct representation for developer engagement using EFA, and validates the method. The next step, in Chapter 7, is to continue with the EFA to find factor structure within the set of metrics for the definition of software health.

Chapter 7

EFA Part II

> *Exploratory factor analysis is like being a detective: You're sifting through the evidence (observed variables) to reveal the unseen culprits (latent variables) behind the scenes.*
>
> unknown

As Informed by the Definition of health from Section 3.2.2, the hypothesised constructs of the EFA are: engagement, interest, and robustness. This chapter picks up where Chapter 6 leaves off in search of latent factors that capture the behaviour of interest and robustness.[1]

7.1 Introduction

The preliminary setup and conditions for the factor analysis is the same as in Chapter 6. The objective of the EFA (Stage I) is to find structure within a dataset (**R**-type) rather than clusters (**Q**-type) as discussed in Section 6.2. Stage II presents the design including variable selection (Section 6.3). The factors are at least just-identified meaning there are indicator variables, $x_1 \ldots x_i, i \geq 3$ (see Figure 4.4). The sample size has the same baseline as for engagement derived in Section 6.3.4, $n = 393$, however, this is reduced slightly due to missing data in newly gathered indicator variables. This analysis begins with selecting indicator variables to represent interest and robustness.

7.2 Interest Metrics

Table 3.2 in Chapter 3 details metrics for engagement and interest. The metrics for interest can be grouped as shown in Table 7.1.

Table 7.1: Detailed metrics for interest popularity grouped into four preliminary categories.

Interest

Category	Description	References[†]
Forks	Count of number of times the software is forked	[4,5,6,7]
Stars	Count of total stars on GitHub; also tags or watchers	[5,6,7,8]
Dependencies	The number of software dependencies a project has, for example, Bitcoin relies on the GCC compiler collection	[7]
Rank	Ranking of the project in the broader web, for example, number of search engine hits, or Alexa page ranking; also downloads	[1,2,3,4,7,9]

[†] Source literature: [1] Crowston et al. (2006), [2] Ghapanchi (2015), [3] Goggins et al. (2021), [4] Jansen (2014), [5] Negoita et al. (2019), [6] Osman and Baysal (2021), [7] Saini et al. (2020), [8] Tamburri et al. (2019), [9] Wahyudin et al. (2007).

The first two items: forks and stars, are the same metrics from Section 6.3.5, but do not contribute to the factor called engagement, and strongly load onto a second factor, undetermined

1. The main result of this chapter is submitted for publication (Nijsse & Litchfield, 2023b).

at the time in Chapter 6. (See Model A in Table 6.7.) For the next two items: dependencies and rank, new data is collected.

Stars The total count of stars received since creation. The same as described on Page 119.

Forks The total number of times a repository is forked since creation. The same as described on Page 119.

Dependencies A project with a large number of dependencies is indicative of users' interest in building with that project. This is closely associated with the number of downloads. Neither of these two measures, dependencies or downloads, are readily available for the blockchain projects so an alternative proxy is used. The Open Source Software Foundation (OSSF) produces a project criticality score (used next, in Section 7.3 for robustness), and here a metric called the number of mentions is used. Mentions is a metric to gauge what projects are popular among contributors and is based on a count of the number of times a project appears in the text through comments of the commit messages. A Python script is written to access GitHub data through the API via the Criticality Score[2] command line tool (version 1.0.7).

Rank The CoinMarketCap API can return a primary and secondary website associated with each project. This website info is manually verified and then used with Amazon's Alexa API to get a global web ranking called Alexa Traffic Rank.[3] A lower traffic rank indicates higher popularity. For example, the highest ranked blockchain related website is `https://www.binance.com/` at 110.[4] A script is written in Python to interact with the CoinMarketCap and Alexa API and retrieve rank data.

7.3 Robustness Metrics

Table 3.3 lists sixteen metrics from the literature conceptually related to software robustness. These can be condensed into eleven groupings shown in Table 7.2. Of these eleven, five are out of scope of the present study with regards to data collection and resource constraints of time and expense of gathering business survey data. The remaining six are collected to assess factor structure related to software robustness.

2. `https://github.com/ossf/criticality_score`
3. Run by Amazon's subsidiary Alexa Internet Inc., the service was shuttered on May 01, 2022.
4. `google.com` is ranked number 1.

Table 7.2: Metrics relating to robustness in the OSS health literature. Many robustness measures are difficult to gauge or subjective in nature such as code quality and knowledge creation. The metrics are grouped into six categories for data collection and five categories that are out of scope for the analysis.

Robustness

Category	Short Description	References†
Developer longevity	Time spent contributing to a single project; also project age	[1,2,5]
Geographic distribution	Global geographic distribution of the contributors	[7]
Market share	Ratio of a project's share to the total local ecosystem	[3]
Project criticality	Risk associated with project centrality and dependency; also truck factor	[2,3,7]
Response time‡	Time between issues being raised and closed; also bug fix time, count of issues closed	[4,6]
Last Updated	Time since the project's codebase has been updated. Stagnant projects are still be accessible but have no sign of activity.	–
Business metrics*	Including management, process development, and systems development; also switching costs	[2,3]
End user metrics*	Including count, longevity, loyalty, and satisfaction	[2,3,9]
Contributor metrics*	Including centrality, reputation, satisfaction, cross org participation; also measures of centrality in wider SECO and partnerships	[3,8]
Code quality*	As relates to code metrics such as cyclomatic complexity	[2,4]
Knowledge creation*	Knowledge added to SECO and artefact creation	[3]

* Metric is considered out of scope and not collected in the present study.
‡ Response time is carried over from the engagement EFA although has literature references to being categorised within robustness as well as engagement.
† Source literature: [1] Chengalur-Smith et al. (2010), [2] Goggins et al. (2021), [3] Jansen (2014), [4] Negoita et al. (2019), [5] Osman and Baysal (2021), [6] Raja and Tretter (2012), [7] Tamburri et al. (2019), [8] Wahyudin et al. (2007), [9] Z. Wang and Perry (2016).

Developer Longevity The average number of days the developers are involved in the project. A unique contributor is determined by username and the time delta in days between their earliest activity and latest (most recent) activity. All the unique contributor's days active info is averaged to get a single value for a project. A larger value indicates that developers tend to stick around, whether contributing continually or sporadically to a project, leaving the project more resilient due to the experience of the contributors. Data is collected from the custom database with a SQL query.

Geographic Distribution This is introduced as a measure of robustness from Table 3.3. This is derived from timezone data retrieved from git history using Perceval (Dueñas, Cosentino, Robles, & González-Barahona, 2018) (version 0.17.0) via a Python script. The timezone data represents the times of software commits made by the project's contributors and produces a mapping of activity based on coordinated universal time (UTC). To evaluate a project's geographic distribution, it is compared to a median distribution of the top 100 blockchain projects over the previous six months. This median distribution is a representation of the typical geographic distribution of the top 100 blockchain projects in terms of software commit activity. The comparison is done by calculating the root mean squared error (RMSE), which is a measure of the difference between the project's geographic distribution and the median distribution. Projects with a low RMSE have a distribution that matches the community and are less prone to geographic shocks, which refer to unexpected events that could disrupt the project's contributors' ability to work together. Projects with a high RMSE likely indicates a project operates in a single timezone and could exhibit single point of failure risks, which refer to the risk that the project's development could be disrupted if a key contributor is unable to work.

Market Share The CoinMarketCap API data provides a rank based on total market of a given project which can be a proxy for financial resources. The ranking is used rather than the numerical value as there is considerable debate about market capitalisation as a measure of blockchain project value because tokens can be artificially inflated. A lower rank indicates more financial resources to be resilient in turbulent times.

Project Criticality The OSSF has a number of metrics that combine to produce a project criticality score. The criticality score is a metric that identifies how critical a project is within the open source ecosystem (Arya et al., 2022). Scored in the range [0,1] a project of score 0 relies on no external software, among other factors, while a project of score 1 is deemed critically important. For example, the highest criticality project overall is Linux while the highest criticality blockchain project is Bitcoin. A Python script is written to access GitHub data through the API via the Criticality Score (version 1.0.7) command line tool .

Response Time Measured in both median and average number of days. The median number of days for issues to be closed based on the GitHub workflow which enables users to open issues and project moderators to close them. Most issues are bug fixing in nature, but can also be feature requests or other activity. A low median time to close issues indicates a reactive community that can withstand shocks and overall a more robust codebase. The average number of days for issues to be closed is also calculated

to gauge potential difference with median response time. Data is collected from the custom database with SQL queries by the process outlined in Section 6.3.3.

Table 3.2 lists bug-fix time as one of the most cited characteristics of OSS engagement. This indicator does not emerge in the EFA for engagement last chapter as it is thought to be inclusive within the common activities of commenting, committing, and pull-requesting. Here it is carried over to be included in the present robustness grouping.

Last Updated The number of months since the project has been updated. Older projects without update can be thought of as stagnating, even simple projects that are not rolling out new features have to occasionally be checked on to update dependencies and bring in line with new standards. There is no official declaration of a project dying, but a project without update in 12 months is considered de facto stagnant. For example, the max in the dataset has not been updated in 70 months and so is likely not going to be robust in the face of ecosystem shocks. A lower updated value is positively valenced; the min in the dataset is 0 indicating updates within the past month. Although not in the OSS health literature, this metric is analogous to an organism staying fit to be able to handle environmental shocks. If the project has not been updated there is less chance it can withstand disruption.

A number of these metrics could be placed in the Interest basket. For example if community members are responsive to requests and updating the codebase this could be seen both as low bug-fix time and as being interested in the project. The benefit of an exploratory factor analysis is that there is no concept of it being *correct* with the bucketing of metrics into specific categories. This is a practical exercise. Beginning with the literature, some place bug fixing in engagement, others place it in robustness, others still prefer the term survivability. What the EFA does is group indicators together that represent, as a bundle, a high correlation within the whole group. Thus, regardless of the bucket naming, the researcher evaluates the factors post-hoc and only then commits to a naming representation.

7.4 DataSet

The dataset is available on GitHub[5] and contains data for 384 blockchain projects on which this chapter's EFA is based. The descriptive statistics are shown in Table 7.3.

5. https://github.com/millecodex/phd

The dataset for Engagement's EFA in Section 6.3.5 has 393, projects, while this one now has $n = 384$. The difference comes from the new metrics that are collected using new data collection methods looking for a repository as listed in the prior dataset that could not be found. The repository has likely moved in the time between data collection and so to maintain temporal integrity these projects are now excluded. Nine projects are in this category.

Missing Data

The dataset under investigation contains missing data for 15 projects, spanning three categories: last updated, mentions, and criticality score. Given that these missing values represent a minor 3.9% of the entire sample (15/384), implementing a data imputation strategy is justified. Two additional projects have a missing Alexa Rank.

The chosen method for this task is mean substitution, a common and widely recognised technique for dealing with missing data (Hair Jr. et al., 2014). This method entails substituting missing values within a variable with the mean value of that variable, as calculated from all valid responses. The underpinning rationale for this approach hinges on the premise that the mean is the most suitable single replacement value.

Notwithstanding its widespread use, mean substitution is not without its drawbacks. First and foremost, it tends to underestimate the variance estimates, given that the mean value is used to replace all missing data. Second, the actual distribution of values may be distorted due to the substitution of the mean for the missing values. Furthermore, the observed correlation can be dampened as all missing data are replaced with a single constant value (Hair Jr. et al., 2014). Despite these limitations, the utility of mean substitution as an approach lies in its ease of implementation and its ability to provide complete information for all cases. As such, it provides a pragmatic solution to the relatively minimal amount of missing data within this study.

7.5 Stage III: EFA Assumptions

The third phase of the EFA involves establishing the underlying assumptions, mirroring the process outlined in Section 6.4 regarding expectations of structured data. To guarantee data homogeneity, the focus is on a specific industry subset within the GitHub ecosystem. The selected subset must be sufficiently extensive to satisfy the required sample size, yet sufficiently specific to adhere to resource limitations.

The statistical premises employed prior to the execution of the factor analysis encompass assumptions regarding the data distribution and the evaluation of the correlation matrix.

TABLE 7.3: Descriptive statistics for 384 OSS blockchain projects used in EFA to identify latent constructs of interest and robustness.

n=384	Mean	Std.Dev.	Min.	Q1	Median	Q3	Max.
Forks	539.64	3437.34	0	12.50	60.00	206.50	59 013
Stars	790.21	4399.46	0	24.00	107.00	413.50	72 112
Mentions	2509.33	26 945.39	0	0.00	15.00	156.00	492 320
Criticality	0.35	0.18	0.02	0.20	0.37	0.49	0.85
Last updated*	6.63	10.66	0.00	0.00	2.00	8.00	64
CMC rank*†	271.15	169.63	1	120.50	260.50	409.00	600
Geographic dist.*	0.36	0.05	0.15	0.33	0.38	0.40	0.47
Longevity	191.32	131.39	0.00	101.02	178.14	256.38	763.80
Alexa rank*	296 412.46	552 446.14	110	43 913.50	131 243.50	290 192.00	4 628 993
Median resp. time*	18.97	29.56	0.00	0.76	2.54	22.95	75.74
Average resp. time*	30.28	36.61	0.00	5.87	12.43	33.80	100.12

* These metrics are negatively valenced where the smaller value has a positive association.
† CMC rank is the ranking from CoinMarketCap

Data Distribution

The distributions continue to be highly non-normal. Figure 6.3 shows distributions for the new metrics: mentions, criticality, updated since, CMC rank, geographic distribution, longevity, Alexa rank, median response time, and average response time.

Standard distribution plots for the additional indicator variables are in Figure 7.1 followed by logarithmic distribution plots in Figure 7.2.

Correlation Matrix Suitability

The correlation matrix (Figure 7.3) applicability for EFA is tested using Bartlett's test of sphericity and the KMO measure of sampling adequacy. First, the negatively valenced metrics in Table 7.3 are re-scored so that the larger values have positive association. Bartlett's test provides a single value for correlation of the data within the matrix, the result of $\chi^2(384) = 2742.981$, at a significance of $p < 0.000$ indicating the data meets the requirement for EFA ($p < 0.05$).

As Bartlett's test is sensitive to sample size, the KMO test is also run. Here the overall KMO is 0.66 and the general rule is that each indicator variable should be above a 0.5 threshold. A value lower than this indicates the data is too random for EFA. Table 7.4 shows the factor adequacy values.

Having met the statistical requirements and assured of the conceptual assumptions, the procedure can now continue to factor derivation.

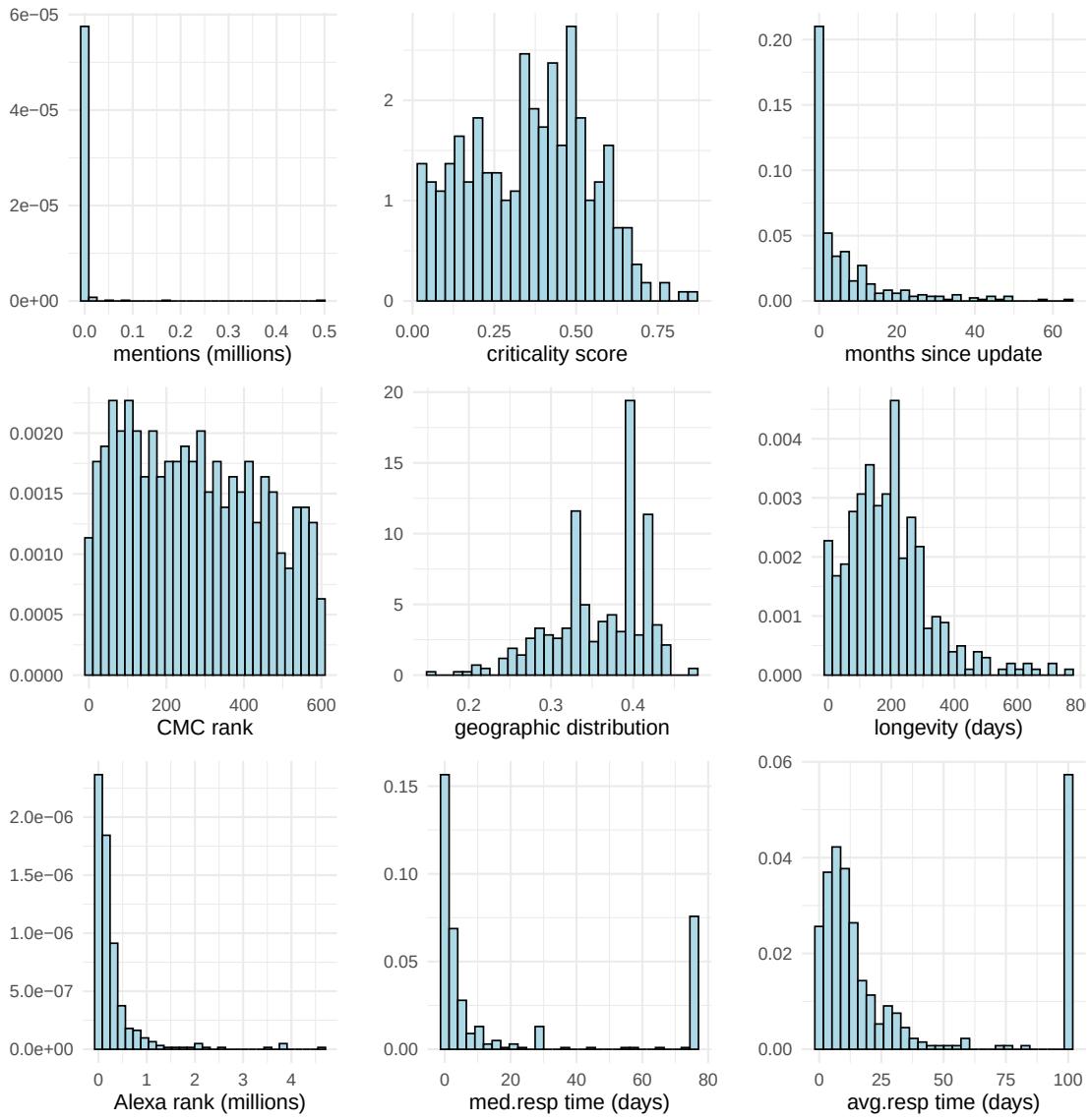


FIGURE 7.1: Histogram Distributions of new metrics from the dataset. The y-axis represents the density, which is the proportion of observations within each bin. Most variables are highly skewed and non-normal. Stars and forks can be seen in Figure 6.3 (Page 123).

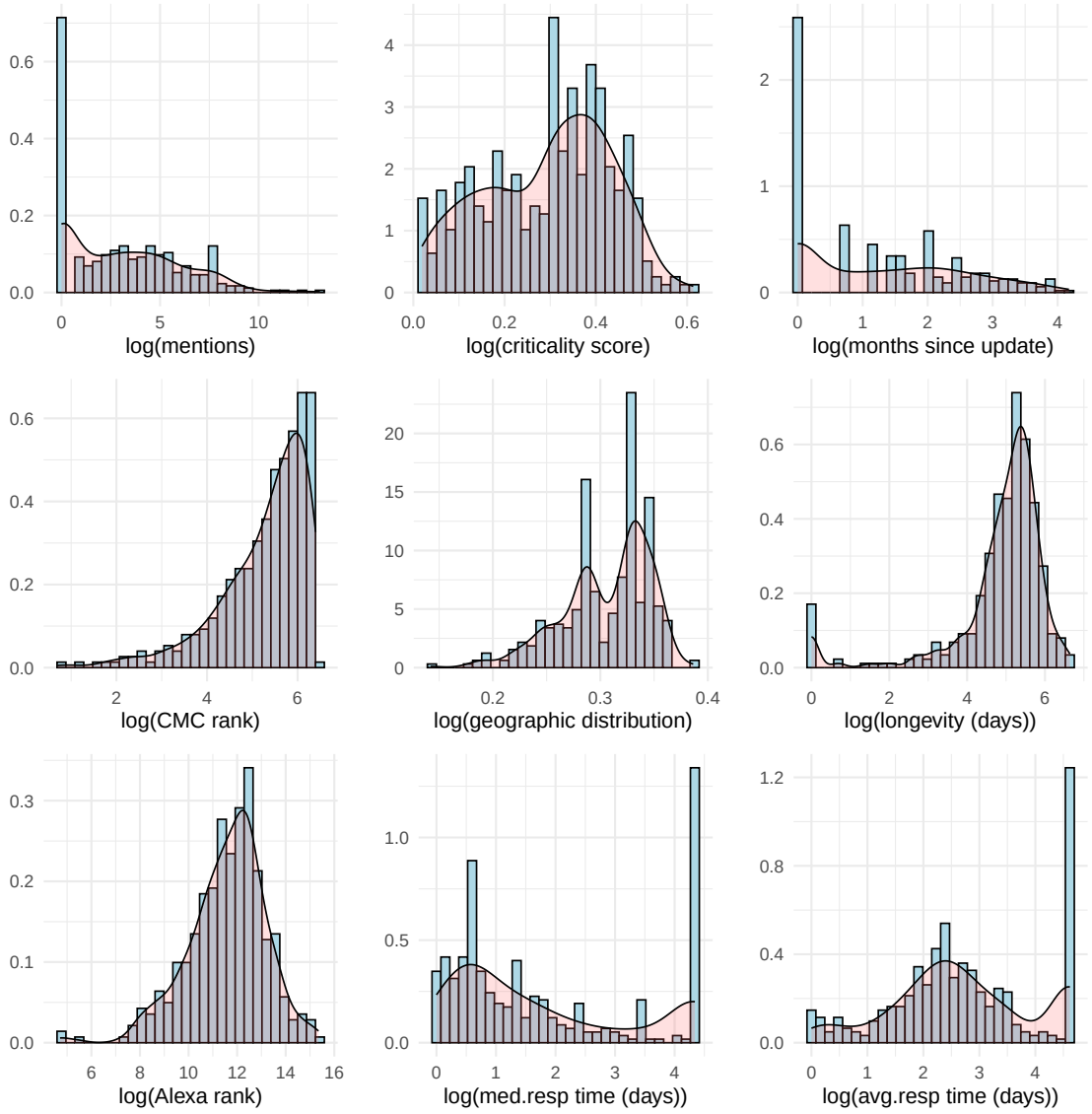


FIGURE 7.2: Logarithmically scaled histogram distributions of the data. The y-axis represents a probability density function for a continuous random variable. The density plot is a smoothed version of a histogram estimated from the data. Stars and forks can be seen in Figure 6.4 (Page 124).

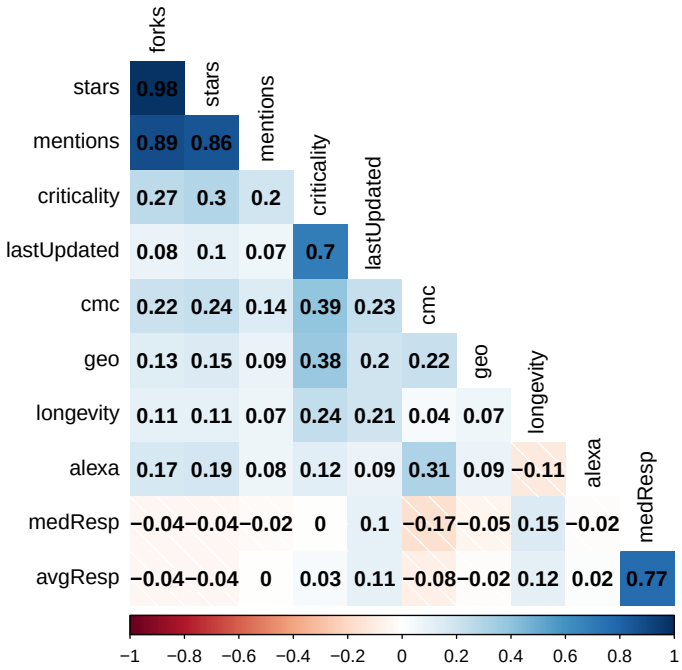


FIGURE 7.3: The Pearson correlation matrix for the dataset shows many strong correlations as input data for EFA. The strongest correlation is 0.98 for stars-forks.

7.6 Stage IV: Factor Derivation

Common factor analysis is applied here with the goal of identifying structure, beginning with determining the number of factors to extract.

How many factors to extract?

Starting with the data in Table 7.3 a scree plot and parallel analysis are examined in Figures 7.4 and 7.5 to see the number of proposed factors. Both a standard scree and parallel analysis indicate preference for two factors with eigenvalues $\lambda_1 = 2.95$ for the first factor and $\lambda_2 = 1.11$ for the second. A parallel analysis provides more robust reasoning as it calculates the eigenvalues of the observed data and compares them to the eigenvalues of randomly generated data. Significant deviation means there is grounding for grouping by factors. The simulated groups using parallel analysis have $\lambda_1 = 0.51$, and $\lambda_2 = 0.23$.

Two- and three-factor analyses are completed to assess the diminishing returns of increased factors.

TABLE 7.4: Kaiser-Meyer-Olkin test scores for individual variables in the dataset. Except for median response time (0.46) all variables meet the 0.5 cut-off for EFA. The overall MSA is 0.66.

Factor	Value
Stars	0.69
Forks	0.65
Alexa rank	0.63
CMC rank	0.77
mentions (dependencies)	0.86
geographic distribution	0.77
criticality score	0.64
longevity (days)	0.75
median response time	0.46
average response time	0.51
updated since (months)	0.59
Overall	0.66

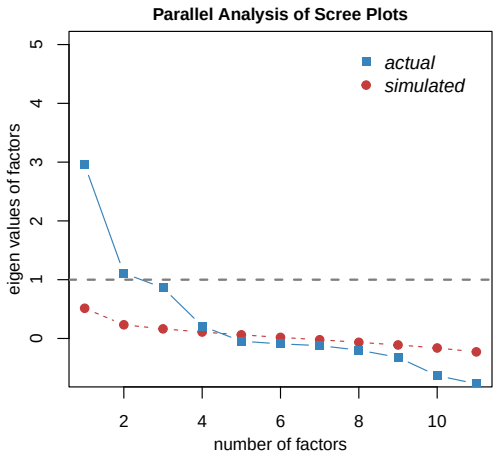


FIGURE 7.4: The parallel analysis shows a clear preference for one factor (eigenvalues > 1) with a mild preference for a second and possibly third factor at the scree-elbow.

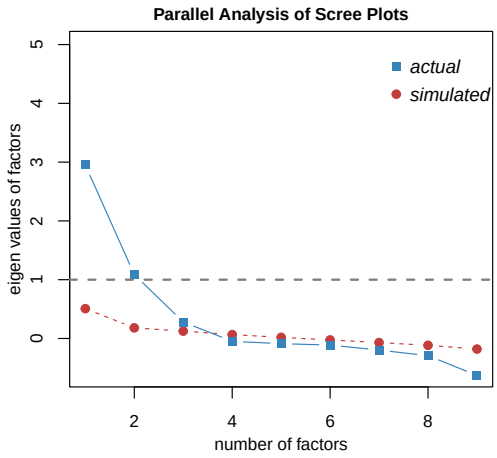


FIGURE 7.5: Removing median and average response time indicators from the dataset leaves a clear indication of two factors present (above dotted line).

7.7 Stage V: Factor Interpretation

Interpreting the factors involves assessing the factor matrix by three steps: first, estimation of the factor matrix method discussed in Section 6.6.1 , second, the rotation method (see Section 6.6.2), and third, number of factors to extract.

7.7.1 Factor Matrix Interpretation

Using the `psych` package factor analysis is completed estimating factors via the maximum-likelihood method. Factor rotation is done with the `GPArotation` package, now selecting the VARIMAX factor rotation method to maximise the variance of the loadings within the factors as it is helpful for structure within two or more factors.

A first iteration of EFA is carried out and both average and median response time exhibit a mild negative influence on the first latent variable—ML_3 in Table 7.5. They both have a strong influence on the second latent variable ML_4, dominating the other variables with the next largest loading being 0.172 for longevity. If response time to resolving issues and fixing bugs is correlated with indicators such as criticality, or how recently the repository is updated, there would be a clearer association with them on ML_4. This suggests that the indicators of median and average response time stand on their own as separate indicator.[6]

Thus, average and median response time are now excluded as part of the EFA iteration process.

Model Respecification

The BIC is a comparator between models, and the BIC improves significantly from 276.252 (in ML_3 & ML_4) to −36.737 from the exclusion of median and average response time from the analysis. Table 7.6 shows the updated EFA factor structure.

7.8 Stage VI: Factor Validation

Model validation is by two mechanisms. First cross-validation of the EFA by randomly separating the dataset into a training and testing segment and comparing model structure. Secondly, confirmatory factor analysis is applied to the measurement model next in Chapter 8.

Cross validation is necessary to avoid the situation where the model ends up being overfit to the data, affecting generalisability. Constraints in the data collection process on the number

6. A 3-factor EFA is done showing just this, however as the two indicators nearly measure the same thing it is unjustified to include these as a further under-identified factor.

TABLE 7.5: EFA loadings for two latent factors ML_3 and ML_4, and common variance, h^2. Strong indicator relationships are in bold. The variables median response time and average response do not fit the factor structure and stand out on their own, dominating ML_4. This is a good indication they do not have relation to the other indicator variables.

Indicator	ML_3	ML_4	h^2
Forks	**0.986**	0.148	0.995
Stars	**0.973**	0.144	0.967
Mentions	**0.875**	0.145	0.788
Criticality score	0.270	0.048	0.075
Last updated	0.069	0.117	0.019
CMC rank	0.241	−0.124	0.074
Geographic distribution	0.139	−0.026	0.020
Longevity	0.084	0.172	0.037
Alexa rank	0.176	0.016	0.031
Median response time	−0.190	**0.979**	0.995
Average response time	−0.155	**0.754**	0.593
SS loadings	2.939	1.653	
Cumulative variance	0.267	0.417	
Proportion explained	0.640	0.360	

TABLE 7.6: EFA loadings for two latent factors ML_5 and ML_6, and common variance, h^2. Strong indicator relationships (> 0.3) are in bold. The variables longevity and Alexa rank do not exhibit enough influence to be included in either latent construct.

Indicator	ML_5	ML_6	h^2
Forks	**0.988**	0.137	0.995
Stars	**0.970**	0.166	0.968
Mentions	**0.885**	0.076	0.790
Criticality score	0.135	**0.988**	0.995
Last updated	−0.015	**0.705**	0.498
CMC rank	0.169	**0.373**	0.167
Geographic distribution	0.082	**0.369**	0.143
Longevity	0.075	0.237	0.062
Alexa rank	0.163	0.104	0.037
SS loadings	2.787	1.868	
Cumulative variance	0.310	0.517	
Proportion explained	0.599	0.401	

of available projects limit collecting a new dataset and so the original is split into two groups. Random allocation is performed using the `caret` package to produce two groups, one to build the model and one to test the model. The split is chosen to allow for half the data to be at the minimum sample size threshold of 200.

Table 7.7 shows the validation results. The insignificant loadings are not shown for readability and to highlight that the same factor structure is present across the models. The training and testing models are both as well fit as the hypothesised model based on χ^2, TLI, and RMSEA, with slight deviations being acceptably close considering the sample size limitation.

TABLE 7.7: Cross-validation for the model showing equivalent factor structure for the testing model as compared to the baseline EFA. Insignificant loadings (< 0.3) are not shown except where appropriate for structure comparison.

	Model		Training		Testing	
Factor	ML_5	ML_6	Tr_1	Tr_2	Te_1	Te_2
Forks	0.988		0.983		0.964	
Stars	0.970		0.976		0.962	
Mentions	0.885		0.924		0.910	
Criticality		0.988		0.994		0.980
Last updated		0.705		0.707		0.711
CMC rank		0.373		0.425		0.292
Geographic distribution		0.369		0.398		0.342
Longevity		0.237		0.288		0.197
Alexa rank	0.163		0.07		0.419	
n	384		213		171	
χ^2	2373.7		1680.1		938.8	
TLI	0.953		0.972		0.969	
RMSEA	0.089		0.077		0.067	

7.9 Analysis: Interest & Robustness

At this stage the latent constructs can officially be renamed to be representative of the underlying indicator variables, thus ML_5 becomes Interest, and ML_6 becomes Robustness. The diagram of the factor structure is shown in Figure 7.6.

With EFA, all measured variables are related to every factor by a factor loading estimate where -1 is a strong negative, 0 is neutral, and $+1$ is strong positive relationship. The significant loads for each factor are shown in boldface along with their loading on the secondary factor. There is strong support for a first factor of Forks, Stars, and Mentions, with a second factor of Criticality Score, Last Updated, CMC Rank, and Geographic Distribution. The chosen

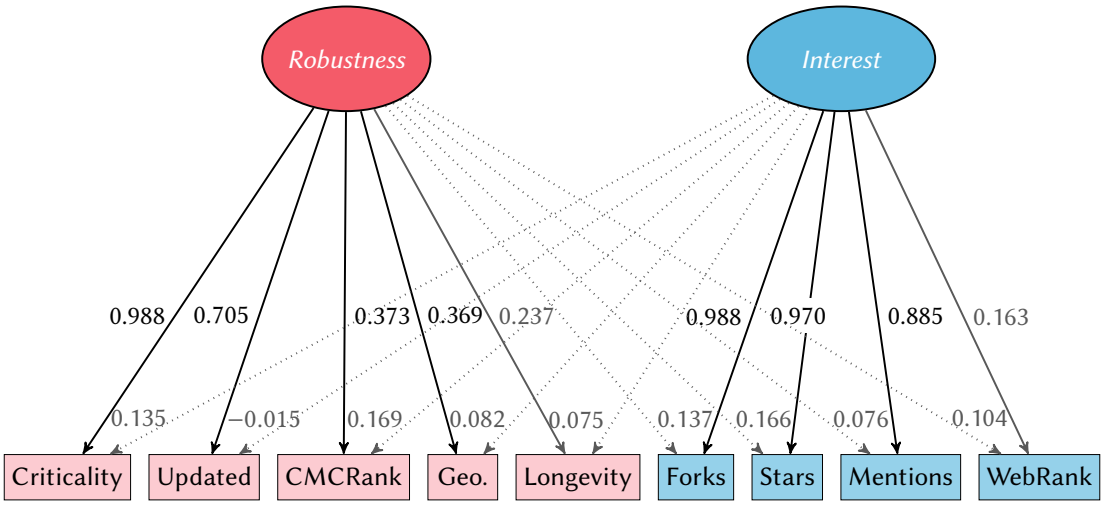


FIGURE 7.6: EFA applied to metrics representing *Robustness* and *Interest*. The primary loadings are the solid arrow in the first tier; the secondary loadings the dotted arrows in the second tier. Note: *median* and *average response time* have already been excluded from the model. Longevity (0.237) and Web Rank (0.163) are excluded at this stage.

cut-off point for loading estimates is 0.3, and thus Longevity is out of range and does not have enough influence to describe either factor. The same applies to Alexa rank, both of which are excluded for the measurement model (Chapter 8).

The communalities (h^2 in Table 7.6) or common variance refers to the proportion of variance in the observed variables that is explained by the corresponding latent factor. It represents the proportion of the variance in the observed variable that can be attributed to the factor, after accounting for measurement error. For instance, Stars has an h^2 of 0.968 for ML_5, indicating that 96.8% of the variance in stars data can be explained by the first latent factor in this model.

SS loadings represents the amount of variance in the observed variables that is accounted for by each factor. Here, the SS loadings for ML_5 and ML_6 are 2.787 and 1.868, respectively, indicating that ML_5 accounts for 2.787 units of variance in the observed variables, and ML_6 accounts for 1.868 units of variance.

The *Cumulative variance* refers to the total amount of variance in the observed variables that can be explained by the factors up to that point. For two factors in the data over half of variance (0.517) is accounted for in the model with 59.9% of it coming from ML_5. The remaining 40.1% is from ML_6.

The EFA diagram in Figure 7.6 shows strong loadings in black and secondary loadings in grey. This model exhibits fit statistics in the range of standard thresholds: the TLI = 0.953, and RMSEA index = 0.089. See Table 7.7 for these in context of the model validation. More on fit statistics is discussed in Section 8.7.

7.10 Conclusion

After validating the method of using exploratory factor analysis in Chapter 6 to determine a factor structure for developer engagement, the present chapter continues in the similar manner to find the factor structure for software robustness and general interest. Figure 7.7 shows the resultant factors representing Robustness and Interest.

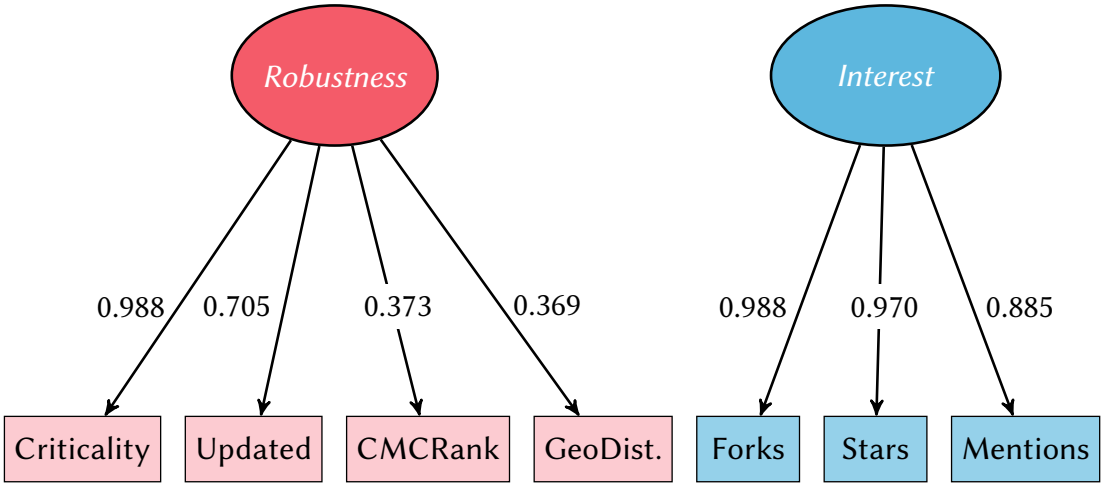


FIGURE 7.7: After iterative EFA processes, two factors are determined with standardised factor loadings shown from the indicator variables.

The new factors required extending the dataset to include metrics for popularity rankings through CMC ranking and Alexa's global web ranking. Additionally, metrics for geographic distribution are calculated, and metrics pertaining to open source software criticality are used. The primary method to validate the present EFA is to complete a structural equation model and is done next, in Chapter 8.

Chapter 8

Structural Equation Modelling for Health

> *All models are wrong, but some are useful.*
>
> George E.P. Box
> 1979

Using the Dataset and factors determined from Chapters 6 and 7, a structural model is proposed and tested.[1] Section 4.3.3 in Chapter 4 outlines how SEM is applicable. This method is particularly relevant to Hypotheses 2b (Figure 4.3), which suggests that (i) Public interest generates developer engagement, and (ii) Developer engagement leads to robust software.

8.1 Introduction

Confirmatory factor analysis is a statistical method used to test the validity of a measurement model in which the relationships between observed variables and their underlying constructs are specified a priori. In other words, CFA is used to confirm the hypothesised structure of a measurement model, rather than discovering it, as in EFA. The SEM process is visualised in Figure 8.1.

CFA is often used in the context of SEM, which is a more general framework for modelling relationships among latent and observed variables. In SEM, a measurement model is specified to represent the relationships between observed variables and their underlying constructs, and a structural model is specified to represent the relationships between latent constructs themselves. The overall goal of SEM is to test hypotheses about these relationships and to estimate parameters that represent the strength and direction of the relationships.

Compared to regression analysis, CFA and SEM are more flexible and can handle more complex models with multiple latent variables and observed measures. CFA allows for the testing of the fit of the hypothesised measurement model, whereas regression assumes that the relationship between the predictor and outcome variables is linear. CFA tests the fit of the hypothesised measurement model by comparing the observed covariance matrix of the variables to the covariance matrix implied by the hypothesised model.

To do this, CFA estimates the factor loadings, which represent the strength of the relationship between the latent variables and the observed measures, and the residual variances, which represent the amount of measurement error in the observed measures. These estimates are used to calculate the covariance matrix implied by the hypothesised model.

CFA then uses a goodness-of-fit statistic, such as the chi-square test, to compare the observed covariance matrix to the implied covariance matrix. The chi-square test evaluates whether the observed covariance matrix is significantly different from the implied covariance matrix, given the degrees of freedom and sample size.

Some benefits of using CFA and SEM include the ability to test complex hypotheses about the relationships between latent and observed variables, the ability to control for measurement error, and the ability to examine the fit of the model to the data. Additionally, SEM

1. The main result of this chapter is submitted for publication (Nijsse & Litchfield, 2023b).

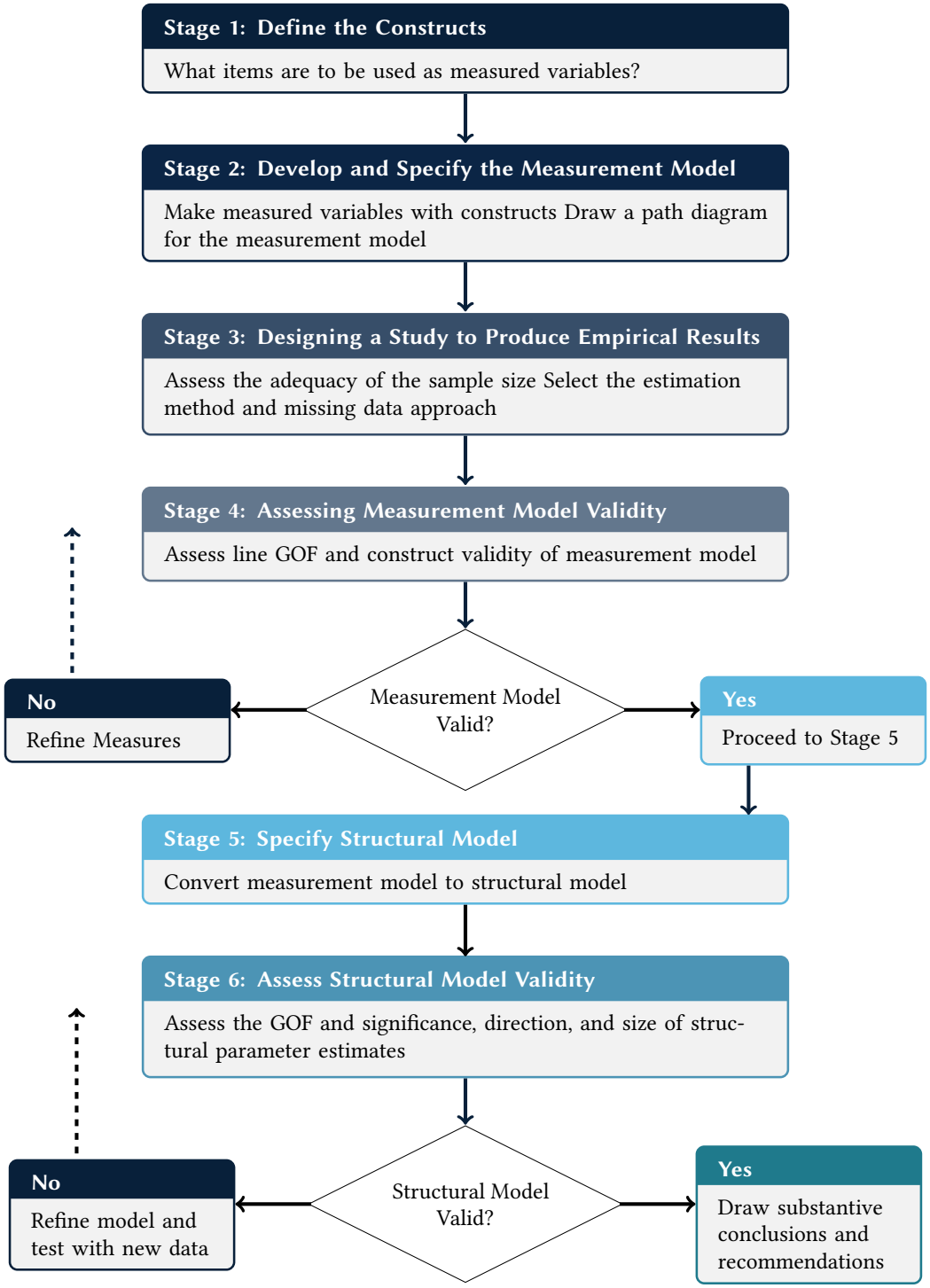


FIGURE 8.1: Six-stage SEM process diagram beginning with construct definition. Two decision points arise where the model can be respecified based on criteria. Adapted from Hair Jr. et al. (2014).

allows for the testing of mediating and moderating effects, which can help to better understand the mechanisms underlying the relationships between variables.

8.2 Stage I: Defining Latent Constructs

The first stage of SEM involves defining individual constructs, or latent variables, and selecting appropriate measurement scales. The constructs are deduced from the literature review and the EFA: general interest, developer engagement, and software robustness. The operisationalising of the constructs is selecting metrics appropriate for representation of the construct, and is generally accomplished through the exploratory phase, as in the present work. The quality of the measurement model plays a fundamental role in the reliability and validity of SEM results. If there are well-defined scales in the literature, the researcher uses these as the basis, or shown in Chapter 3, new scales for health are selected. Chapters 6 and 7 are the pretesting ground for defining new metrics and operationalising all indicators. Once latent variables, and indicator variables, and measurement scales are in order, the measurement model is specified.

8.3 Stage II: Specifying Measurement Model

In the stage of specifying the measurement model, the researcher identifies the latent constructs and assigns the measured indicator variables (items) to each construct. There are three types of relationships to be specified: measurement relationships between indicators and constructs, structural relationships between constructs, and correlational relationships between constructs. Figure 8.2 illustrates the basic measurement model with three constructs, with one being just-identified with three indicators, and two being over-identified with four indicators. There is also a correlational relationship between the constructs.

This measurement model is derived from the exploratory work that started in Chapter 6 where indicators are found for engagement. Continuing into Chapter 7 that found indicators for interest and robustness. Presently, the model in Figure 8.2 is hypothesised to be a representation of the data collected (Section 7.4).

SEM employs a reflective measurement theory, which posits that latent constructs are the underlying causes of the observed variables. Any discrepancies between the latent constructs and the measured variables are attributed to measurement errors, preventing a complete explanation of the observed variables. Consequently, the arrows in the model diagram (Figure 8.2) depict the relationship from latent factors to measured variables. This reflective measurement approach aligns with the principles of classical test theory (Hair Jr. et al., 2014).

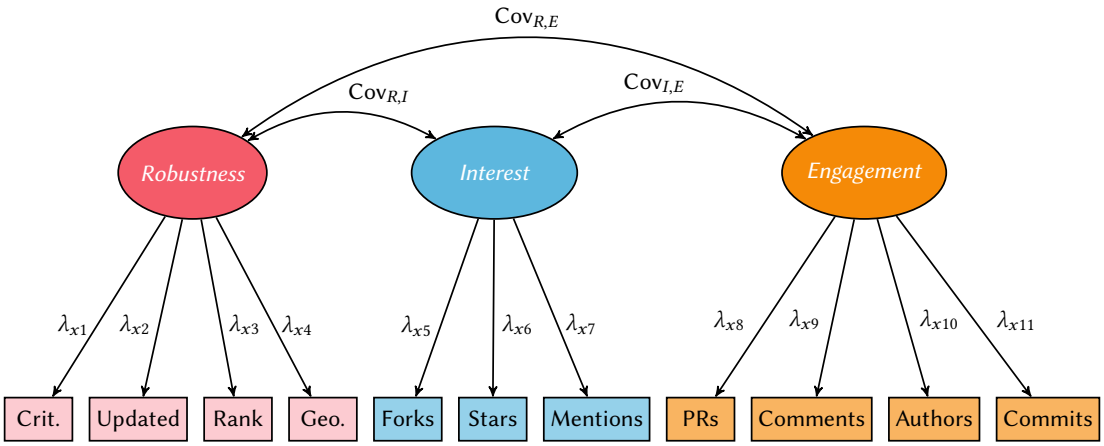


FIGURE 8.2: Hypothesised measurement model (path diagram) for OSS health combining constructs of engagement, robustness, and interest. Loadings are represented by λ_{xi}, covariance by Cov, and error terms are not shown.

8.4 Stage III: Design Decisions

Decisions in the study design process are completed as part of the EFA, and inherited in this section. These include methodology and reasoning to handle missing data, sample size considerations, data type and normality, matrix methods, and model estimation. The relevant details are summarised in Table 8.1.

TABLE 8.1: The design decisions as the basis for the SEM models. The selection and rationale is inherited from the EFA process.

Design Decision	Selection	Rationale
data type	metric	Table 7.3
data distribution	highly non-normal	Section 7.5
sample size	384	Section 6.3.2
missing data method	mean substitution	Section 7.4
matrix input	correlation	Section 7.5
model estimation	ML	Section 6.6.1

Where necessary, these design decisions are revisited, such as for respecification of the model.

8.5 Stage IV: Measurement Model Validity

At this stage the measurement model is tested using CFA to compare the proposed model to the dataset. The CFA analysis is carried out with the `lavaan`[2] package (version 0.6.13) in **R** (version 4.0.2). There are no cross-loadings in the CFA since it limits the loading to the theoretical construct only, setting any remaining loadings to zero.

Model Definition

The measurement model in Figure 8.2 is translated into model syntax for `lavaan` in Figure 8.3, where latent factors are defined with =~ and the combination of indicator variables separated with +.

```
CFA-1 <-
# measurement model latent factors
  interest   =~ forks + stars + mentions
  robustness =~ criticality + lastUpdated + cmc + geo
  engagement =~ authors + prs + comments + commits
```

FIGURE 8.3: CFA-1 the measurement model defined in Section 8.3 written in `lavaan` syntax to be estimated in CFA. The three latent variables interest, robustness, and engagement, are defined with =~ syntax as a combination of the indicator variables.

8.5.1 First Iteration Results

The output of the CFA applied to the measurement model above estimates the covariance matrix. This estimation tells the strength of the chosen latent factors influencing the selected indicator variables. The model fit compares the estimated covariance matrix to the observed covariance matrix. A large deviation between these two indicates the model does not represent the reality that informed the dataset. A small deviation says the model may be a representation of the data and therefore a generalisation of reality. The `lavaan` results are shown in Table 8.2.

The model fit parameters are shown in Table 8.3. These are included for completeness and for comparison, as the presence of Heywood cases is the primary motivation for reassessing the model structure.

The estimation of CFA1 yields two slightly negative variances: forks at −0.002 and criticality at −0.002. Negative variances, also referred to as Heywood cases in factor analysis,

2. In Chapter 6 the `psych` package is used for the EFA, here the `lavaan` package is used because the model syntax carries over to SEM.

TABLE 8.2: `lavaan` results for model CFA-1 with standardised loadings > 0.3 shown and ordered decreasing by factor loading.

Indicator	Interest	Robustness	Engagement
Forks	1.000		
Stars	0.979		
Mentions	0.887		
Criticality score		1.000	
Last updated		0.694	
CMC rank		0.390	
Geographic distribution		0.375	
Comments			0.967
Pull requests			0.882
Authors			0.870
Commits			0.677
Avg. variance extracted	0.917	0.444	0.732
Construct reliability	0.733	0.640	0.746

occur when the model estimates a variance (or a covariance) that is less than zero. While this is mathematically possible in the context of model estimation, it does not make sense in real-world terms because variances (and covariances) cannot be negative (Hair Jr. et al., 2014).

The appearance of negative variances usually indicates that there may be issues with the model or the data including model misspecification, multicollinearity, and inadequate sample size. A small negative variance might suggest that the model is nearly correct but slightly overparameterised or the data is almost adequate but has minor issues, such as slight multicollinearity. In contrast, a large negative variance might indicate a more substantial issue with the model or data, such as a serious misspecification or a significant multicollinearity problem.

The appearance of Heywood cases, albeit slight, is motivation for model respecification.

8.5.2 Model Re-Definition

A number of options are available to achieve improved model estimation and may be implemented as long as they also exhibit strong theoretical backing. Options include fixing loadings to a predetermined value, matching loadings between indicators, freeing paths to allow for multi-factor relations, removing problematic indicators, and modification indices (Hair Jr. et al., 2014; Willem E. Saris & van der Veld, 2009; Rosseel, 2020). This is depicted in Figure 8.1 when the measurement model does not meet validation requirements.

A correlation path between forks and stars is freed to improve the model by eliminating a

TABLE 8.3: Fit statistics for model CFA-1. Listed here primarily for comparison purposes as the presence of Heywood cases dictates the model to be respecified.

Model	CFA-1
χ^2	597.655
DoF	41
p-value	<0.001
CFI	0.840
TLI	0.786
BIC	8432.183
RMSEA	0.210
SRMR	0.104

potential Heywood case. A negative error value is illogical as it means that more than 100% of the variance in the data is due to the factor structure. The alternate remedy here is to remove forks altogether as a metric, however, it remains for two reasons. First, because including forks allows for three indicators on interest (dropping to two indicators is considered underfit), and second it is an important concept in OSS, and as shown in Figure 8.14 possibly the most important metric in the dataset. The correlation between forks and stars has grounding in the literature (Abdulhassan Alshomali, 2018; Osman & Baysal, 2021), and empirically in the dataset as the Pearson correlation (Figures 6.5 and 7.3) is 0.98.

```
CFA-2 <-
# measurement model latent factors
  interest =~ forks + stars + mentions
  robustness =~ criticality + lastUpdated + cmc + geo
  engagement =~ authors + prs + comments + commits

# correlations
  forks ~~ stars
```

FIGURE 8.4: Measurement model CFA-2 in `lavaan` is redefined from CFA-1 with the addition of a correlation, `~~`, between forks and stars.

Second Iteration Results

Table 8.4 shows the standardised loadings from CFA-2. The Heywood cases form CFA-1 are eliminated and the model can now be evaluated. The path diagram with loadings is shown in Figure 8.5.

The average variance extracted for Robustness (0.444) is slightly below the 0.5 rule indicating that, on average, more than half of the variance is not explained by the factor structure.

TABLE 8.4: `lavaan` results for model CFA-2 with standardised loadings > 0.3 shown and ordered decreasing by factor loading.

Indicator	Interest	Robustness	Engagement
Forks	0.966		
Stars	0.943		
Mentions	0.917		
Criticality score		0.998	
Last updated		0.694	
CMC rank		0.391	
Geographic distribution		0.375	
Comments			0.970
Pull Requests			0.878
Authors			0.865
Commits			0.674
Avg. variance extracted	0.890	0.444	0.731
Construct reliability	0.728	0.640	0.745

Robustness still contributes, but not as clearly the other constructs.

The construct reliability for interest and engagement is above the 0.7 threshold indicating that the items represented share a high proportion of the variance in common. Robustness has a construct reliability of 0.640 which sits in the grey area between 0.6 and 0.7 indicating that it may be acceptable when considering the other factor's value (Hair Jr. et al., 2014). As robustness is the only of three factors below the threshold, and below by a small 0.06, it will stay in the model.

The data shows good support for the hypothesised model indicating the latent constructs are represented by their indicator variables. However, the goodness of fit statistics in Table 8.5 indicate there is room for model improvement. The CFI is 0.84 and the TLI is 0.78 both of which are under a 0.90 heuristic threshold, and the RMSEA is 0.214 (>0.07) and SRMR is 0.106 (>0.08), both of which are over heuristic thresholds meaning there is evidence for good model fit, but not great. These fit statistics must be used cautiously as there is evidence that with sample sizes approaching 400 the maximum likelihood estimator becomes sensitive to changes in the data resulting in poor fit (Hair Jr. et al., 2014) as evidenced by the statistically significant p-value (<0.001) that translates into poor model fit. In this chapter the p-value is reported for completeness, however, the significance is not of concern in the context of the non-normal maximum likelihood estimation. Chi-square values are reported using the Yuan-Bentler corrected chi-square (Yuan & Bentler, 2000) to handle non-normal data.[3]

3. The Yuan-Bentler is the standard correction factor reported for `lavaan`, Mplus, and EQS software.

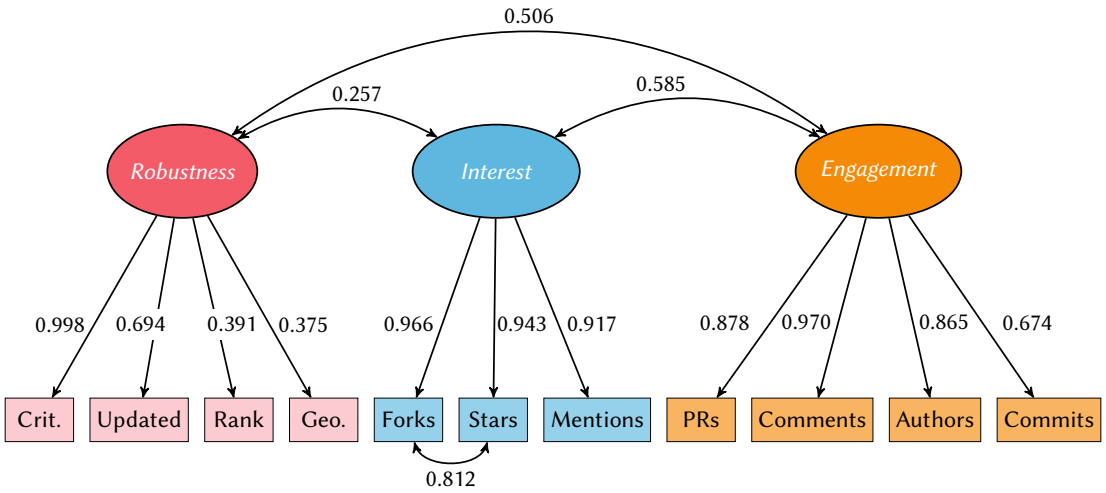


FIGURE 8.5: Model CFA-2 showing correlations between latent constructs and between forks and stars. Loadings are standardised across all eleven indicator variables.

TABLE 8.5: Fit statistics for model CFA-2.

Model	CFA-2
χ^2	629.945
DoF	40
p-value	<0.001
CFI	0.840
TLI	0.780
BIC	8435.670
RMSEA	0.212
SRMR	0.106

Internal consistency reliability is measured to ensure that the items in the scale are measuring the same construct consistently which increases confidence in the validity of the scale. Estimates of internal consistency reliability for each scale based on Cronbach's alpha (Cronbach, 1951) and McDonald's omega (McDonald, 1999) coefficient show high levels of internal consistency reliability for all scales with alpha coefficients ranging from 0.69 to 0.97, indicating good to excellent reliability. The omega coefficients are also high, ranging from 0.73 to 0.95, indicating good to excellent general factor saturation. These estimates suggest that the scales are reliable measures of the constructs they are intended to measure. In other words the data from the selected metrics are accurately captured in each latent variable.

Having met the requirements for a CFA, the second sub-research question in Figure 4.3: What

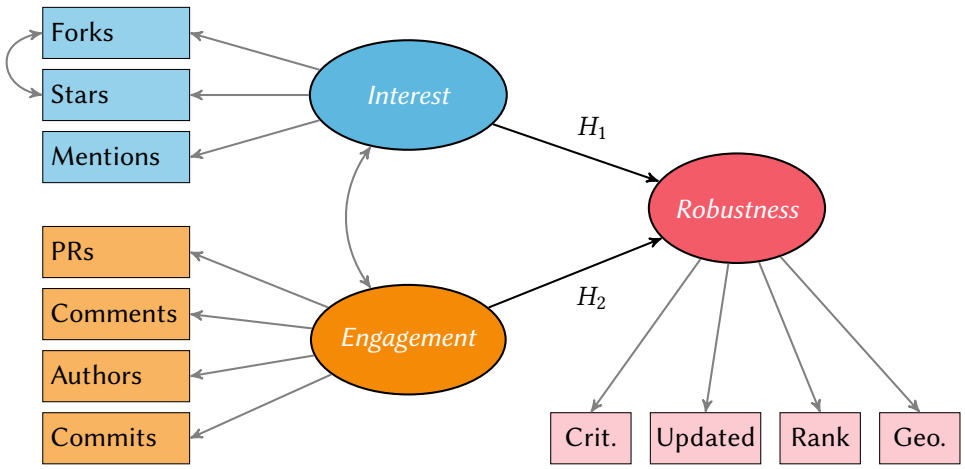


FIGURE 8.6: SEM-1 path diagram showing the hypothesised structural model with path relationships: H_1, and H_2. Correlations are present between forks and stars, and the latent variables interest and engagement.

is the nature of the relationship between factors influencing software health? is investigated with SEM.

8.6 Stage V: Structural Model Definition

The proposed structural model, SEM-1, is derived from the definition of software health (Section 3.1.1) as composed of the three latent constructs found in the EFA: interest, engagement, and robustness. Figure 8.6 shows a structural model with two hypothesised path relationships. The structural part of the model is represented by the single-headed arrows that depict the dependence relationship between constructs. The first hypothesis states:

$$H_1 : \textit{Interest} \rightarrow \textit{Robustness}$$

Put another way: interest positively influences robustness. This relationship captures the behaviour that general interest or popularity of a project leads to greater software robustness. The second hypothesis states:

$$H_2 : \textit{Engagement} \rightarrow \textit{Robustness}$$

This independent path says that developer engagement is positively related to software robustness. The correlation between interest and engagement is retained in Figure 8.6 as a double-headed curved arrow.

Figure 8.6 is the structural model to be estimated by SEM which includes the indicator estimates and the hypothesised path relationships. Figure 8.7 shows the conversion of the graphical representation into lavaan syntax.

```
SEM-1 <-
# measurement model latent factors
  interest =~ forks + stars + mentions
  robustness =~  criticality + lastUpdated + geo + cmc
  engagement =~ authors + prs + commits + comments

# structure
  robustness ~ engagement + interest

# correlations
  forks ~~ stars
```

FIGURE 8.7: Structural model SEM-1 is converted from model CFA2 (Figure 8.4) in lavaan syntax with the addition of path relationships between interest and robustness and engagement and robustness.

8.7 Stage VI: Structural Model Validity

Assessing model validity involves fit statistics and evaluating path coefficients as compared to theoretical hypotheses. This occurs in an iterative fashion where revisions necessitate re-estimation of model weights (see Figure 8.1). Beginning with SEM-1, three iterations are shown presently which lead to the final model, SEM-4.

Model SEM-1 Results

SEM-1 results in Figure 8.8 show a high correlation between interest and engagement (0.585) suggesting there might be an underlying structural relationship between them. As interest represents the lighter touch activities such as starring a repository, it is reasonable to suggest that this activity predicts engagement or more developer-centric activities such as committing to a code repository. The structural relationship is added in the next iteration, revised model SEM-2 in Figure 8.9.

Model fit statistics are summarised in Table 8.6. The values are nearly identical to the fit statistics from model CFA-1 (Table 8.3). This is to be expected because this is a saturated model and the same number of path relationships exist as construct correlations in the CFA. It is noted here that the SEM model cannot improve upon model fit characteristics of a CFA model, however, this is not the intention. The purpose of the SEM is to test the CFA and

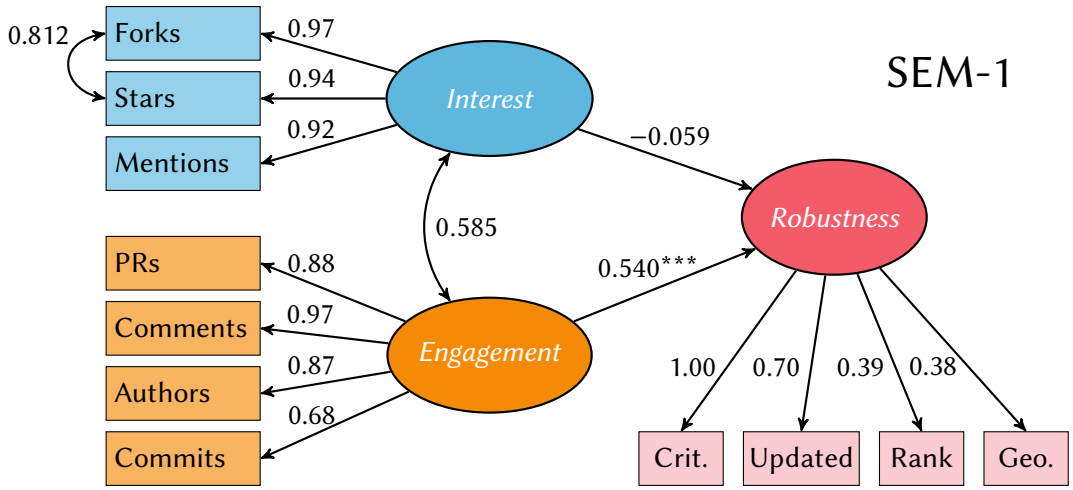


FIGURE 8.8: SEM-1 Results. Path diagram showing factor loadings standardised across all indicator variables, covariance between interest and engagement 0.585, stars and forks 0.812, and structural relationships between interest and engagement and robustness, with standardised path coefficients of 0.540 ($p < 0.001$) and −0.059 (not significant), respectively.

determine levels of support for the hypothesised path relationships. As with the CFA, Yuan-Bentler is reported for χ^2, and robust values for non-normality.

Model SEM-2 Results

SEM-2 represents the addition of the path relationship:

$$H_3 : \textit{Interest} \rightarrow \textit{Engagement}$$

TABLE 8.6: Fit statistics for model SEM-1. These are nearly identical to the model CFA-2 (Table 8.5), which is to be expected.

Model	SEM-1
χ^2	629.941
DoF	40
p-value	0.001
CFI	0.840
TLI	0.780
BIC	8435.670
RMSEA	0.212
SRMR	0.106

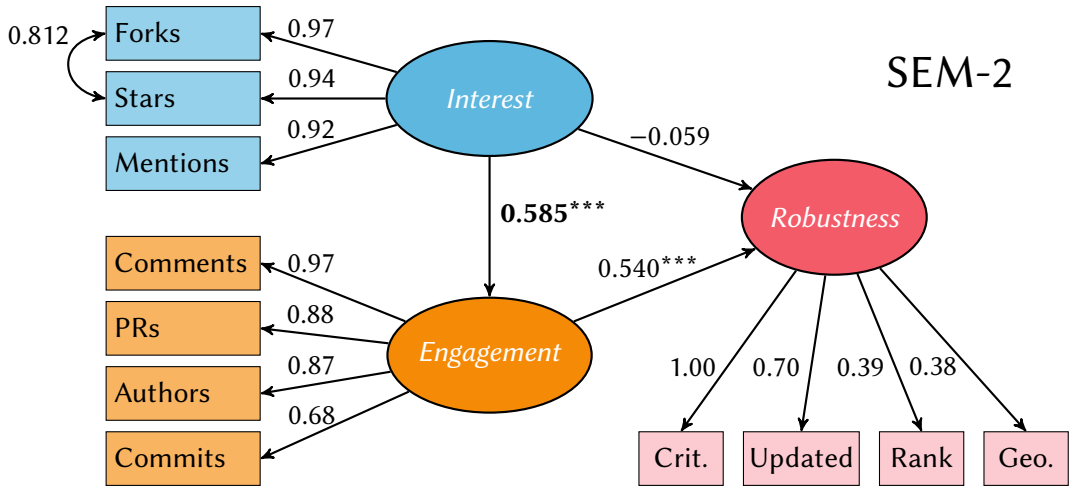


FIGURE 8.9: SEM-2 Results. Path diagram showing the addition of the path relationship: *Interest* → *Engagement* at 0.585 is statistically significant at $p < 0.001$. The remaining standardised loadings are consistent with SEM-1 results in Figure 8.8. .

to SEM-1. The model is re-estimated and shown in Figure 8.9. Here, the path loadings remain the same and there is a clear directional relationship between engagement and interest.

The SEM results in Figure 8.9 show there is almost no effect of interest on robustness (loading of −0.059, $p > 0.05$) and thus this relationship, H_1, is not supported. The second hypothesis, H_2, has strong support that engagement predicts robustness (loading of 0.540, $p < 0.001$). The third result here (H_3) is that general interest is a leading indicator of developer engagement and supported with a loading of 0.585 at $p < 0.001$. The fit statistics for SEM-2 are identical to that of SEM-1 in Table 8.6.

Model SEM-3 Results

Model SEM-3 in Figure 8.10 revises model SEM-2 to remove the path relationship H_1. The results are similar to SEM-2 showing H_2: *Engagement* → *Robustness* strength 0.50, and H_3: *Interest* → *Engagement* strength 0.58. The loadings are nearly identical to SEM-2, given that robustness is not allowed to correlate with interest, the loadings are slightly lower at 0.01 for indicators last updated and geometric distribution.

The variance estimate for the criticality indicator on robustness is slightly negative, at −0.009. The level of caution here is low because the negative variance is close to zero and the fit statistics are not markedly different from SEM-2. However, the presence of a Heywood case is enough to warrant a model respecification. Refer to Section 8.5.1 for a description of Heywood cases. Fit statistics are compared across all four models in Table 8.8.

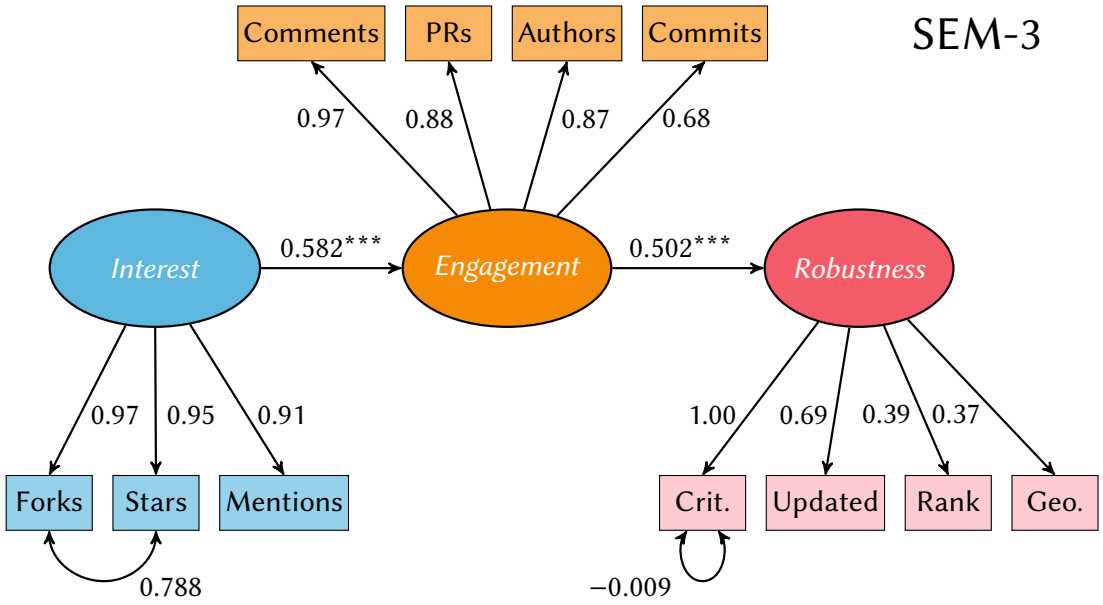


FIGURE 8.10: SEM-3 path diagram after removing the structural relationship between robustness and interest. The remaining relationships are statistically significant at p <0.001, however, the criticality indicator has a negative variance −0.009 motivating respecification to SEM-4.

Model SEM-4 Results

To address the Heywood case estimated in SEM-3 the criticality indicator, which is the strongest loading on robustness for the three SEM models, is set to 0.9, thereby restricting its influence on robustness. Last updated is also fixed to 0.7 to limit its outsized influence. The respecified SEM-4 model is shown in Figure 8.11.

```
SEM-4 <-
# measurement model latent factors
  interest =~ forks + stars + mentions
  robustness =~ 0.9*criticality + 0.7*lastUpdated + geo + cmc
  engagement =~ authors + prs + commits + comments

# structure
  robustness ~ engagement
  engagement ~ interest

# correlations
  forks ~~ stars
```

FIGURE 8.11: SEM-4 defined with loadings fixed for criticality and last updated.

The resulting estimated variance for criticality in Figure 8.12 is now positive (0.053). Loadings for criticality are 0.97, and for updated are 0.73; values that are consistent with previous

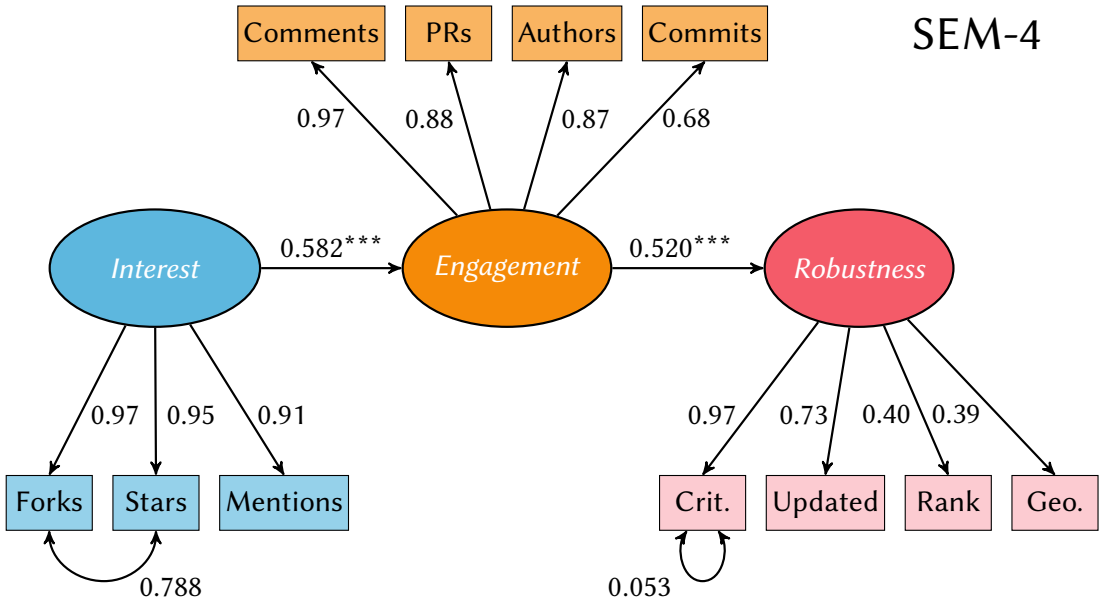


FIGURE 8.12: SEM-4 Results. Path diagram showing revised model after constraints placed on the criticality and last updated indicators.

models.

A method to address negative variances is to set loadings for similar indicators equal. For example, setting criticality and last updated equal also eliminates the negative variance but artificially increases the influence of last updated and decreases the influence of criticality. Given criticality is the strongest indicator of robustness, the loadings are fixed to retain the relative proportion.

A comparison of the loading estimates of CFA-2 with SEM-4 is in Table 8.7. The loading estimates are all very close to their designation in CFA-2 with the furthest deviation coming from the last updated indicator which is fixed when defining SEM-4. The construct reliability has improved slightly placing all factors in SEM-4 above the 0.7 threshold. The average variance extracted for SEM-4 (not shown in Table 8.7) is also similar.

Table 8.7: Model CFA-2 and SEM-4 with standardised loadings and construct reliability shown. *Last updated exhibits the largest differential between models and is due to the fixed loading defined by SEM-4.

Indicator	CFA-2	SEM-4
Forks	0.966	0.973
Stars	0.943	0.950
Mentions	0.917	0.913
Criticality score	0.998	0.973
Last updated	0.694	0.730*
CMC rank	0.391	0.403
Geographic distribution	0.375	0.387
Authors	0.865	0.866
Pull requests	0.878	0.881
Comments	0.970	0.970
Commits	0.674	0.676
Construct reliability		
Interest	0.728	0.728
Robustness	0.640	0.717
Engagement	0.745	0.816

Table 8.8: Fit statistics for models SEM-1, SEM-2, SEM-3, and SEM-4 for inter-model comparison purposes. There is a slight improvement from SEM-1 to SEM-4.

	SEM-1	SEM-2	SEM-3	SEM-4
χ^2	629.941	629.947	584.347	593.248
DoF	40	40	41	42
p-value	<0.001	<0.001	<0.001	<0.001
CFI	0.840	0.840	0.841	0.840
TLI	0.780	0.780	0.787	0.791
BIC	8435.670	8435.670	8430.662	8427.650
RMSEA	0.212	0.212	0.209	0.207
SRMR	0.106	0.106	0.106	0.106

Fit statistics for all four SEM models are compared in Table 8.8. These statistics only show slight improvement from the description in Table 8.5 and are here for inter-model comparison purposes. Chi-square is a measure of the difference between the observed and expected covariance matrices, and improves between SEM-1 and SEM-4, however, it's sensitive to sample size.

The p-value is reported for completeness, however the insignificance is not of concern in the context of non-normal ML estimation. TLI is still under the 90 heuristic, however, it improves from SEM-1 to SEM-4. BIC reduces in the same interval, albeit only slightly. RMSEA comes down slightly in SEM-4 and SRMR remains unchanged as expected because the SEM structure cannot improve these fits.

8.8 Analysis

In general, a measure is considered to have good nomological validity if it is consistent with the theoretical principles and empirical evidence in the field. This means that the test scores should be associated with other measures in the expected direction, and that the relationships should be consistent across different populations and contexts. Nomological validity is an important aspect of test validation because it helps ensure that the measure is accurately measuring the construct of interest. The literature has already informed the factors, and so this test is a conceptual check on the outcome relationships. After removing the hypothesised path between interest and robustness (H_1, Section 8.6), there are two remaining paths:

Interest → Engagement The interest construct is made up of forks, stars, and mentions which are closer to social metrics than traditional software development. For example, someone that is interested in Bitcoin is likely to first star the repository as a bookmarking method, then fork it if they are further curious. These activities happen before any discussion about bugs or proposed changes, and before any new code is written and a pull request submitted. So, in the procedural sense of contributing to OSS, interest leads, or predicts, engagement.

Engagement → Robustness Going back to the ecology metaphor, an organism must be able to sustain its base metabolic needs for survival before it can grow and thrive in its environment. Only once the basic needs are met can it become strong enough to survive shocks and adapt to changes in the environment. In software, this base survival is the day-to-day operations and involves communicating with community members, writing code, submitting code reviews, and attending to comments. If these needs are met through the contributors' motivation and satisfied working conditions, then the project can be in a position to strengthen against unknown future disruptions.

Reframing the structure from left to right per the structure in SEM-3 and SEM-4 helps

with understanding of inputs to the software process as interest, which informs engagement, which again, in sequence, loads on robustness. Figure 8.13 shows the latent factors in an orderly manner which conceptually contributes to the nomological validity of the model and underpins the framework in Chapter 9.

FIGURE 8.13: Structural relationship of latent factors: left to right showing natural order.

8.9 Validation

Validity of the structural model is assessed in Section 8.7 and involves assessing the statistical measures of model fit as well as the nomological validity of the result. Here, two additional external methods are used to test the data against the theoretical constructs. First, bootstrapping is applied to the dataset and the model is estimated based on new, bootstrapped values. Second, a Bayesian Networks (BN) approach is taken to search for structured paths in the dataset.

8.9.1 Bootstrapping

Bootstrapping is a statistical technique where data is resampled with replacement from an original dataset to create numerous alternative samples, referred to as bootstrap samples. This strategy proves particularly useful when estimating the precision of sample estimates such as variance or standard deviation, especially when the underlying data distribution is unknown or complex. In SEM with the `lavaan` package in **R**, bootstrapping assesses the stability and validity of the model parameters to obtain robust standard errors and confidence intervals for these estimates. By simulating the sampling distribution through resampling, bootstrapping in SEM assists in validating the model and providing robustness checks for parameter estimates, thereby enhancing the reliability of conclusions drawn from the model.

The bootstrapped standard errors are comparable to the model fit standard errors. Bootstrapped samples of size 500, 1000, 2000, and 5000 are computed and shown in Table 8.9.

Bollen-Stine bootstrapping is used to account for robust non-normality in the data (Bollen & Stine, 1992). The method first transforms the data such that the null hypothesis holds exactly in the resampling space. This method accounts for model misfit when creating the bootstrap samples, which may result in more accurate inference about model parameters. By including

TABLE 8.9: The bootstrapped chi-square and path estimates for the Bollen-Stine bootstrapping method as compared with the Model SEM-4 with upper and lower confidence intervals (ci, 95%) for the paths. *standard error differential is largest for path *H3*.

samples	500	1000	2000	5000	SEM-4
Test Statistic					
χ^2	744.249	744.249	744.249	744.249	744.249
dof	42	42	42	42	42
p value	0.078	0.085	0.094	0.088	0.000
Path H2: Engagement → Robustness					
coefficient	0.646	0.646	0.646	0.646	0.646
std. error	0.133	0.145	0.140	0.147	0.124
p value	0.000	0.000	0.000	0.000	0.000
ci.lower	0.348	0.321	0.326	0.315	0.404
ci.upper	0.870	0.888	0.876	0.892	0.889
supported	yes	yes	yes	yes	yes
Path H3: Interest → Engagement					
coefficient	0.518	0.518	0.518	0.518	0.518
std. error*	0.436	0.501	0.444	0.472	0.043
p value	0.235	0.301	0.243	0.273	0.000
ci. lower	−0.528	−0.673	−0.539	−0.617	0.434
ci. upper	1.181	1.291	1.202	1.235	0.602
supported	no	no	no	no	yes

the model misfit in the resampling process, the Bollen-Stine method can provide more accurate confidence intervals for the parameter estimates, especially when there is substantial misfit between the model and the data.

The Bollen-Stine bootstrap procedure does not change the test statistic itself, rather, it alters the distribution that the test statistic is compared against in order to compute the p-value. What the Bollen-Stine method does is to simulate a new sampling distribution of chi-square statistics based on the observed data, under the assumption that the SEM-4 model is correct. This simulated distribution takes into account the specific characteristics and potential violations of assumptions in the data.

The p-value from the Bollen-Stine method is then calculated as the proportion of chi-square statistics in this simulated distribution that are larger than the original chi-square statistic. This can be very different from the p-value calculated under the assumption of perfect multivariate normality, which is what the original chi-square test assumes. Thus, the bootstrapping procedure supports the model definition to be representative of the underlying dataset.

The 95% confidence intervals in Table 8.9 provide good support for the path *Engage-*

ment → *Robustness* with significant $p < 0.001$ lending support to the statement that engagement influences robustness.

The exception is the third hypothesised structure H_3: *Interest* → *Engagement* which is not replicated in the bootstrapped model and has a large standard error compared with estimated model. Thus, the overall model robustness when considering bootstrapping is split, with path *H2* showing support and path *H3* not being supported.

8.9.2 Bayesian Network Path Analysis

Bayesian networks offer a statistical framework for modelling complex systems under uncertainty. These graphical models provide a compact, intuitive, and flexible way to represent the joint distribution over a set of variables. They accomplish this by capturing the conditional dependencies between variables through a directed acyclic graph (DAG), where nodes correspond to variables and directed edges indicate dependencies (Cooper & Herskovits, 1992). By representing these dependencies graphically, BN facilitate a clear visualisation for mapping theory to data.

BN path modelling extends this framework to examine the paths of influence among variables in the network. The direction of the edges in the DAG allows the tracing of paths from one variable to another, providing insights into the possible chains of influence between them. Both direct and indirect dependencies between variables are captured where they are appropriate. Determining and analysing these relationships is via Bayesian Network Path Analysis (BNPA). Through its capability of structure discovery, BNPA is leveraged to learn potential causal structures directly from data by identifying probable paths of influence among variables.

BNPA is a structure learning approach, where the aim is to discover the underlying structure of the data without strong prior assumptions. It's analogous to EFA but with a focus on the direct dependencies among variables rather than latent structures. This is similar to how EFA allows every indicator variable to cross load on every other indicator variable. The application of BNPA to a dataset post priori can bolster or weaken the SEM results. BN analysis has demonstrable use in areas such as risk assessment and human reliability analysis (Zwirglmaier, Straub, & Groth, 2017), environmental and ecosystem science (Marcot & Penman, 2019), and software defect prediction (Okutan & Yıldız, 2014).

BNPA is implemented in the **R** package `bnlearn` (version 4.8.1) and displayed graphically in Figure 8.14. The hill climb heuristic search optimisation algorithm (Russell & Norvig, 2009) is used to fit a network structure to the dataset (Section 7.4). Nodes in the DAG are coloured according the latent variables defined in model CFA-2 (Figure 8.5). As there is no concept of latent constructs in BN DAGs any emergent clustering is due to the structure learning

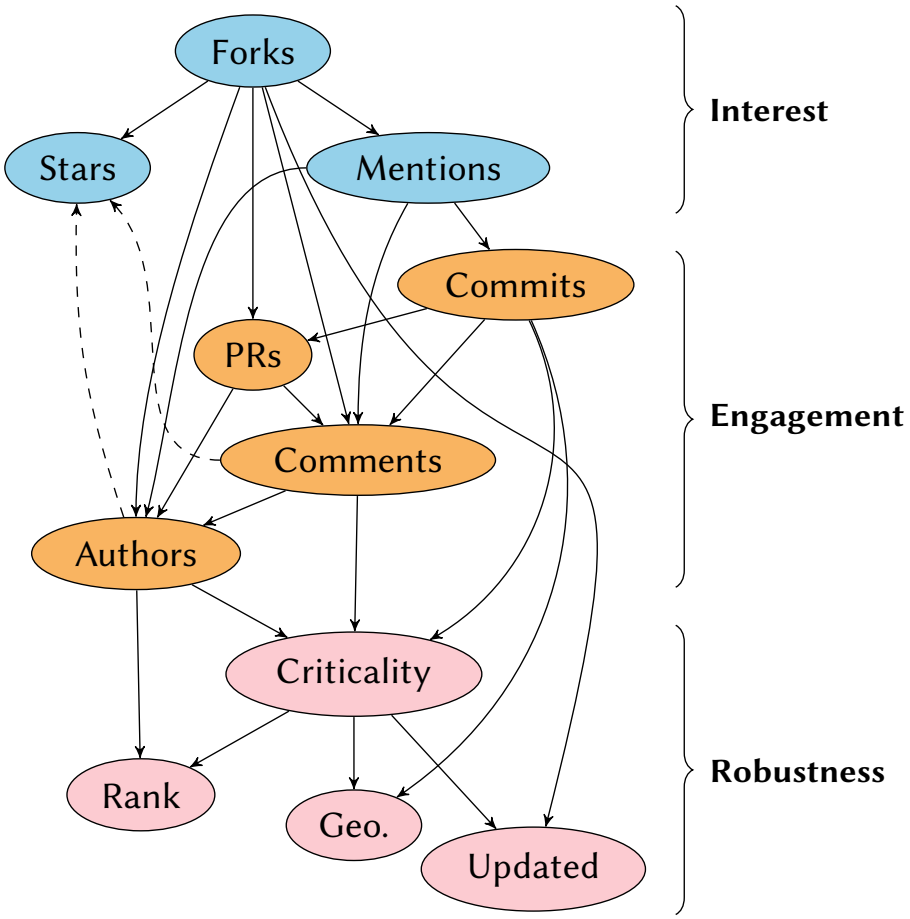


FIGURE 8.14: Bayesian network DAG produced from the dataset with no predefined whitelisting or blacklisting using a hill climb optimisation algorithm. Each directed edge is a statement of influence. When grouped according to latent factors, all paths except two proceed from root to leaf downward in the modelled direction: *Interest* → *Engagement* → *Robustness*. This structure corresponds to that seen in Figure 8.13.

approach. In this case the root of the DAG is forks (within interest), the bulk of the activity in the middle is encapsulated in the engagement factor, and the children nodes are the robustness metrics.

Further BN analysis is out of scope of the present work but could be beneficial by investigating edge strength and other optimisation algorithms such as min/max optimisation (Tsamardinos, Brown, & Aliferis, 2006) or simulated annealing (Russell & Norvig, 2009).

8.10 Conclusion

Chapters 6 and 7 establish the derivation of latent factors pertaining to interest, engagement, and robustness. This chapter integrates these latent factors into a measurement model, employing CFA and SEM for testing. The model structure is influenced by three hypothesised paths, out of which two are supported with statistical significance (p <0.001), while one lacks support and is removed.

The SEM-4 model is evaluated for its statistical fit, demonstrating slight improvement on previous models. Model validation is accomplished through the application of bootstrapping, utilising the Bollen-Stine method with up to 5000 samples. This bootstrapped model reinforces path H2, which postulates that engagement leads to robustness, while it fails to support path H3, suggesting that increased interest does not leads to increased engagement.

To further the validation of the structural model a BN machine learning analysis is completed on the raw dataset. Without any predetermined structure via whitelisting and blacklisting the BN DAG shows strong directional support for the theoretical model of health.

Though the SEM model's fit statistics and validation could be enhanced for more decisive compliance with statistical thresholds, there is substantial directional evidence indicating that (a) the latent factors are represented in the dataset, and (b) a structural relationship exists between engagement and robustness, whereas there is a lack of correlation between interest and robustness.

These established relationships depicted in Figure 8.13 lay the foundation for the ensuing framework presented next in Chapter 9.

Chapter 9

Theoretical Framework

> *Data are not theory.*
>
> Robert Sutton and Barry Staw
> *What Theory is Not*
> 1995

> *The process of theorizing consists of activities like abstracting, generalizing, relating, selecting, explaining, synthesizing, and idealizing.*
>
> Karl E. Weick
> *What Theory is Not, Theorizing Is*
> 1995

Chapter 8 Uses the Latent factors in Chapters 6 and 7 to develop a structural equation model that is validated by bootstrapping and a Bayesian path analysis. This chapter introduces the development of a theoretical framework built from the model designed to explore and highlight the key constructs and relationships that govern OSS blockchain project health.

9.1 Introduction

The approach within intends to address not only the "what" and "how" of blockchain software health, as developed in RQ2a (Chapters 6 and 7) and RQ2b (Chapter 8), but also the "why," thereby offering a well-rounded framework capable of advancing both theoretical understanding and practical applications. A brief introduction to frameworks is presented before distinguishing between conceptual and theoretical frameworks.

Theory of Frameworks

Defined by Kerlinger (1973, p.9), "A theory is a set of interrelated constructs (concepts), definitions, and propositions that presents a systematic view of phenomena by specifying relations among variables, with the purpose of explaining and predicting the phenomena." Cushing (1990) places the framework as a step along the road to theory, where theory is a single all encompassing goal of the field as a whole. In this sense there is a distinction between a framework and a theory. Gregor (2006) delineates the steps of theory building into various terms, such as classification schema, frameworks, and taxonomies, all of which can be theory in their own right. Within frameworks themselves further delineation occurs between conceptual frameworks and theoretical frameworks (next).

Drawing inspiration from Gregor's classification of theories, the framework incorporates elements typically associated with theory development, such as means of representation, primary constructs, statements of relationships, and testable propositions (2006). The framework presented here begins with a conceptual framework that is built upon to meet the theory requirements such as the power to explain and predict. Further discussion towards a theory of software health is in Section 10.5.

The framework is intended to serve as a tool that simplifies the organisation and understanding of OSS metrics that contribute to the current state of software health serving as a structured lens through which the phenomenon can be examined.. The goal is to help practitioners and researchers answer the following questions:

1. How to structure an open source project to be healthy?
2. How to evaluate and improve current projects for health?

The subsequent sections of this chapter detail the components of the framework, its contributions to knowledge, inherent limitations, and potential applications and implications. By presenting this theoretical framework, the chapter aims to contribute to the ongoing dialogue and research in the open source community, laying the groundwork for further inquiry and innovation in the field of open source blockchain projects.

9.2 Conceptual Framework

Frameworks can be categorised as theoretical or conceptual, each serving a distinct purpose in research. Theoretical frameworks are constructed upon existing theories and models, focusing on explaining or predicting specific phenomena through causal relationships and testable propositions. They often incorporate hypotheses and are geared towards empirical validation. In contrast, conceptual frameworks provide a more descriptive and interpretive view, mapping key concepts and their relationships without necessarily positing causal connections (Jabareen, 2009). Theoretical frameworks aim to provide rigour and specificity through their connection to literature, while conceptual frameworks offer flexibility by not attempting to explain the relationships, thereby allowing hypotheses to be formed (Grant & Osanloo, 2014).

A conceptual framework for OSS blockchain health is presented in Figure 9.1. The purpose is to understand the key factors influencing the health of open source software projects and how they interact. Previously, the key concepts are defined as

- Interest: Measures of community interest, such as forks, stars, and mentions (Section 7.9),
- Engagement: Indicators of developer engagement, including authors, pull requests, commits, and comments (Section 6.8), and
- Robustness: Metrics reflecting the stability and quality of the software, such as criticality, last updated date, and geographic distribution (Section 7.9).

A final concept is now defined that envelops the prior three: Health. An overarching concept reflecting the state of the OSS project, as informed by the latent factors of interest, engagement, and robustness. Health has previously been defined in Section 3.2.2. The conceptual framework identification of health shows it as a concept outside of latent factor definition. Additionally it is not exclusively composed of interest, engagement, and robustness, there is undefined potential for researchers to use the framework in their own exploration and interpretation.

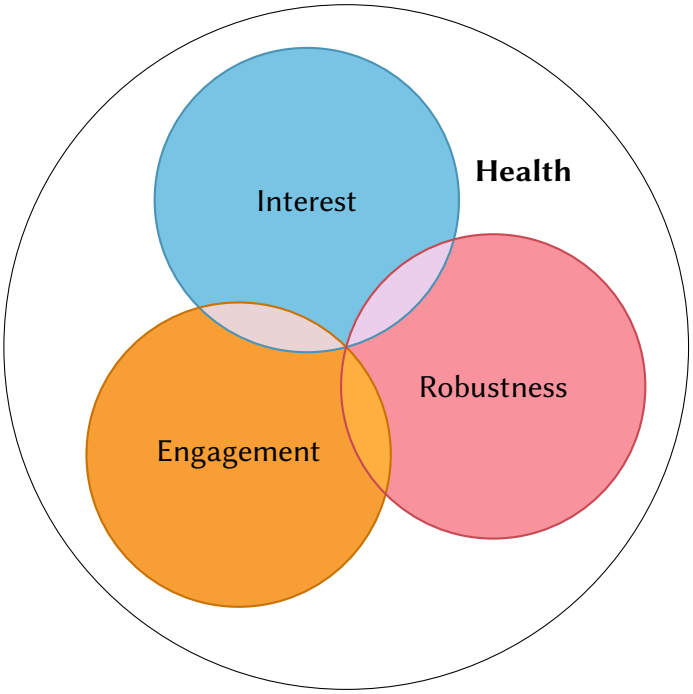


FIGURE 9.1: A conceptual framework for OSS blockchain health showing the key factors and how they interact.

The conceptual framework presents a variety of relationships. Interest ⇔ Engagement: How community interest relates to engagement without positing a specific causal connection. Engagement ⇔ Robustness: How developer engagement relates to the robustness of the software. And Robustness ⇔ Interest. There are also simultaneous relationships: for example, engagement can be related to both robustness and interest. Health is present as a combination of interest, engagement, robustness, but not exclusively composed of these components.

The visual representation is intentionally neutral; this conceptual framework has no explicit directional relationships and therefore is not in a position to offer prediction capabilities. The lack of directional arrows removes the potential that would imply causality. Nor is the health circle split in a recognisable fraction that could indicate one factor has size preference over the other. Lastly, Figure 9.1 is without ordering; no factor can be said to come first in a list.

This conceptual framework satisfies the criterion of flexibility and description (Jabareen, 2009). It is flexible in interpretation, allowing researchers and practitioners to explore the interconnectedness of interest, engagement, robustness, and overall health without being confined to specific causal assumptions. It is also descriptive, offering a view of the key factors influencing OSS blockchain health and how they interact, providing a basis for further explo-

ration and analysis.

But it cannot, however, predict specific outcomes, as it lacks directional arrows or specific causal relationships that would allow for precise predictions or explanations. Additionally it does not allow for empirical validation, since it does not include testable hypotheses or propositions.

The conceptual framework, when combined with the SEM (Figure 8.12), serves as a starting point for the development of a theoretical framework.

9.3 Theoretical Framework

Gregor (2006) outlines seven elements of theory in IS that, when combined, represent a theoretical contribution to the field. The elements are described in turn, and summarised in Table 9.1.

9.3.1 Means of Representation

Means of representation refers to the medium through which the theory is communicated. In the context of this framework, the means of representation include verbal descriptions, diagrams for visual representation, and software for practical implementation. The use of multiple means of representation ensures that the theory is accessible and understandable to a wide range of audiences, including researchers, developers, and users of blockchain technology.

Primary Representation: The core constructs and relationships are visually represented in a diagram (Figure 9.2) that illustrates the flow from general interest to developer engagement and finally to software robustness. The entire diagram symbolises the health of the open source blockchain project, with specific metrics arranged within each construct from top to bottom, reflecting their strength. The top represents the most influential variables, while the bottom houses the weaker ones. Although representative of health, the diagram does not predict it.

Secondary Representation: The textual description that defines the constructs, relationships, and scope. Alongside the diagram, the theory articulates itself through a detailed description, providing clarity and depth to the visual representation.

TABLE 9.1: The blockchain health analysis framework involves seven components that when combined represent a new theory.

Component	Instantiation
1. Means of Representation	Figure 9.2 is the **visual** depiction of the theory which displays relationships, ordering, and concepts through the use of **language** as explained in the present work.
2. Primary Constructs	**General Interest**, defined in Section 3.3.1; **Developer Engagement** Section 3.3.2, defined in; and **Software Robustness**, defined in Section 3.4.
3. Statements of Relationship	Three statements based on Figure 8.11: **(i)** Interest positively impacts developer engagement; **(ii)** Engagement positively impacts robustness; **(iii)** Robustness reflects a state of health. (Section 9.3.3)
4. Scope	**Level of Generality**: the theory is applicable to highly open sourced software industries, with a specific focus on blockchain projects at this stage. **Boundary Statement**: The theory does not apply to business or corporate software development structures. **Modal Qualifiers**: The theory's statements and constructs are relevant to "all" open source blockchain projects within the defined boundaries within the project timeframe. (Section 9.3.4)
5. Causal Explanations	Interest is a requirement for voluntary contributive work in OSS and thus general interest precedes developer engagement. Discussed in Section 9.3.5.
6. Testable Propositions	Three high-level based on Statements of Relationship: **(i)** increased interest results in increased developer engagement; **(ii)** increased developer engagement results in increased software robustness; and **(iii)** robustness is a reflection of project health. (Section 9.3.6)
7. Prescriptive Statements	**For example:** Strengthen developer engagement by focussing on pull-requests and total authors metrics; foster general interest by encouraging forking; note the correlation between forks and stars metrics; monitor criticality and repository updated dates to maintain robustness. (Section 9.3.7)

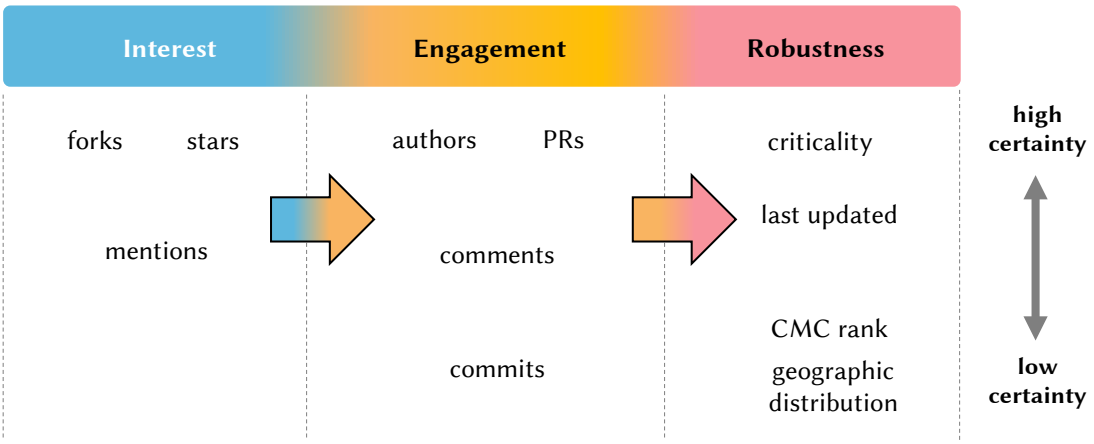


FIGURE 9.2: A theoretical framework for OSS blockchain health. Factors proceed from left to right with individual metrics (as defined in Chapters 6 and 7) ordered from top signalling a higher level of certainty to bottom where metrics have a lower level of strength of influence.

9.3.2 Primary Constructs

Primary constructs are the fundamental concepts or variables that the theory is built upon. There are three primary constructs defined by the present research:

1. **General Interest**: A latent factor representing the collective curiosity, attention, and enthusiasm toward the open source blockchain project within the community. Derived from specific observable variables in the literature (Section 3.3.1), it encapsulates the multifaceted aspects of interest that contribute to software health.

2. **Developer Engagement**: A latent factor reflecting the active participation, collaboration, and commitment of developers in the project. It is constructed from various indicators found in the literature that gauge the level of participatory developer involvement through coding activities and related metrics (Section 3.3.2).

3. **Software Robustness**: A latent factor signifying the resilience, stability, and efficiency of the open source blockchain software. It encompasses observable variables in the literature (Section 3.4) that assess the software's ability to withstand shocks to the operating environment.

9.3.3 Statements of Relationship

In Gregor's framework for theory development, the statements of relationships are fundamental to articulating how the primary constructs are interconnected. These statements define the relationships between constructs, whether causal, correlational, or dependent, and are

expressed through the means above (Section 9.3.1). The complexity of these relationships may vary, encompassing direct or indirect links, reciprocal connections, feedback loops, or hierarchical structures.

The model presents three relationships:

1. **Interest positively impacts developer engagement**: The latent factor of interest, representing the collective curiosity and attention toward the open source blockchain project, exerts a positive influence on developer engagement. This causal relationship signifies that an increase in interest leads to a corresponding increase in developer engagement, reflecting active participation and collaboration within the project.

2. **Engagement positively impacts robustness**: The latent factor of engagement, encompassing active developer participation and commitment, positively influences software robustness. This causal link denotes that heightened engagement causes an enhancement in the resilience, stability, and efficiency of the software, contributing to its overall robustness.

3. **Robustness reflects a state of health**: Strong robustness, stemming from the positive impacts of interest on engagement and engagement on robustness, represents a state of health in the open source blockchain project. This relationship encapsulates the theory's central premise that cumulative effect of interest, engagement, and robustness collectively reflect the health of the software.

These statements of relationships are essential to the theory's goal of explanation and prediction.

9.3.4 Scope of the Theory

The scope defines the boundaries of the theory, including any assumptions or limitations. Clearly defining the scope ensures that the theory is applied appropriately and helps to avoid misinterpretations or overgeneralisations.

For this framework, the scope is explained first through the level of generality, next the boundary conditions, third the modal qualifiers, and last the potential extensions.

1. **Level of Generality**: The theory is applicable to highly open sourced software industries, with a specific focus on blockchain projects at this stage. It encompasses open source blockchain projects at the repository scale across different stages of development, and various community sizes.

2. **Boundary Statement**: The theory does not apply to business or corporate software development structures. Its principles, relationships, and propositions are tailored to the dynamics of open source development and may not translate directly to closed, commercial, or corporate environments. The theory applies to blockchain project repositories and thus does not apply at the larger levels of an organisation or ecosystem.

3. **Modal Qualifiers**: The theory's statements and constructs are relevant to "most" open source blockchain projects within the defined boundaries. Exceptions and variations may exist, reflecting the diverse and evolving nature of open source communities and technologies, however, based on the data collection techniques in Section 6.3.3 the qualifier "most" rather "all" is more apt.

4. **Potential Extensions**: The immediate extensive capability to be tested is beyond the repository level to the organisation level of a project. While the primary focus is on blockchain, the theory's principles may be explored or adapted to other open source software domains, recognising the shared characteristics and challenges of open source development. This is explored in Section 11.4.

9.3.5 Causal Explanations

Causal explanation involves providing explanations for why certain outcomes occur based on the relationships between constructs. These explanations contribute to the explanatory goal of the framework by providing an understanding of why a particular relationship exists or why a certain phenomenon occurs. Causal explanations explore the underlying mechanisms and processes that give rise to observed relationships. Causality is a natural extension of explanation and should be treated with care, but not ignored.

Gregor (2006, p.628) emphasises the importance of causality as a central element in the formation of theory. Regarding researchers in IS describing theories of the present type (explanation and prediction): "it appears authors have not quite made up their mind as to whether causality is allowable in a theory or not, or where it can properly be mentioned." Gregor goes on to say researchers, "should make their commitment clear and couch their propositions in terms that show what they really mean."

An explanatory statement from the theoretical framework is: General interest precedes developer engagement. Transcribed into a causal statement: a high level of general interest causes increased developer engagement. A second statement is: developer engagement leads to better software robustness. And with a causal qualifier: strong levels of developer engagement cause robust software.

Noting that the degree of certainty in the prediction is expected to be only approximate or

probabilistic in IS, these statements offer practical, clear guidance for practitioners. Causality is further discussed in Section 10.5.1.

9.3.6 Testable Propositions

Testable propositions, which are the sixth component within the framework for theory in IS, play a pivotal role, serving as specific predictions that can be empirically evaluated. They often involve hypotheses about causal relationships, adding rigour and credibility to the theoretical construct. These testable propositions provide a roadmap for empirical investigation, allowing researchers to design studies, collect data, and analyse results to validate or challenge the theory's assertions.

The present theoretical framework posits three testable propositions to aid researchers and developers in evaluating OSS blockchain systems.

Proposition 1: An increase in the latent factor of interest, as measured by specific metrics reflective of general community attention and popularity, results in a corresponding increase in developer engagement within the OSS blockchain project.

Proposition 2: An increase in developer engagement, as measured by indicators outlined in Figure 6.9, leads to enhanced software robustness.

Proposition 3: The combined effect of strong robustness, derived from the positive influences of interest on engagement and engagement on robustness, is indicative of the overall health of the OSS blockchain project.

Extending beyond a mere hypothesis, these assertions can be empirically tested and validated (or falsified) by applying the framework across various blockchain systems, thereby providing a concrete avenue for demonstrating the practical value and theoretical soundness of the framework.

Section 10.5.2 expands these propositions into some hypotheses for researchers to further investigate these dynamics in OSS.

9.3.7 Prescriptive Statements

The final theoretical element is prescriptive statements. These provide actionable guidance derived from the theory's principles and empirical evidence. They serve as a bridge between the theoretical understanding and real-world application, offering recommendations for practitioners, developers, project managers, researchers, or other potential stakeholders in OSS and blockchain.

General Prescriptive Statements on OSS Blockchain Project Health

1. To enhance Community Interest: Focus on increasing the number of forks, stars, and mentions. Engage with the community through social platforms, forums, and collaboration channels to enhance these metrics.

2. To strengthen Developer Engagement: Encourage the active participation of authors, facilitate pull requests, reduce focus on code commits, while promoting discussion through comments. Implement inclusive practices, recognition programs, and collaborative platforms to boost these indicators.

3. Build Software Robustness: Emphasise the criticality of issues (with higher weights on criticality), monitor last updated timeframes, and consider geographical diversity to enhance robustness. Implement rigorous testing, quality controls, and continuous improvement processes to align with these factors.

Specific Prescriptive Statements on OSS Blockchain Project Health

1. Track Forking and Starring Practices: Encourage community members to fork projects on platforms like GitHub to enhance visibility and interest. Forking and starring are highly correlated; concentrate on forking codebases as the primary metric for interest.

2. Monitor criticality rankings: a decrease in relative ranking to the industry could mean reduced robustness.

3. Update Repositories Regularly: projects that are updated recently are more likely to exhibit robust behaviour.

4. Foster Global Collaboration: Encourage borderless collaboration to bring diverse perspectives and enhance the robustness of the software.

5. Promote Author Contributions: Recognise and reward authors for their contributions to foster a sense of ownership and engagement.

6. Recognise Correlations between Forks and Stars: Acknowledge the correlation between forks and stars, utilising strategies that promote both, recognising their interconnected influence on interest.

Components one through seven making up the theoretical framework are summarised in Table 9.1. Evaluation of the framework is now analysed.

9.4 Framework Evaluation

The framework evaluation methodology by Weber (2012) consists of a detailed, two-pronged approach focusing on both the individual components (the parts) and the overall structure (the whole) of a theory. Initially, the evaluation concentrates on the quality and clarity of the individual elements that comprise the theory, including constructs, associations, states, and events, ensuring each is well defined and interrelated. Subsequently, the methodology shifts focus to assess the overall coherence, contribution, and applicability of the theory as a unified whole.

9.4.1 Evaluation of the Parts

The evaluation scrutinises individual components, namely constructs, associations, states, and events.

Constructs: The clear and precise descriptions that represent the phenomena within the theory. This is captured in component two in Table 9.1 – Primary Constructs, and the criteria is met by offering clear definitions of the constructs and their interrelations.

Associations: The relationships or connections within the theoretical framework, ensuring they are logical and well-defined. Component three – Statements of Relationship in Table 9.1 captures directional influence (Figure 8.11) for the theory. Additionally, the SEM study provides empirical evidence within the boundary conditions (Section 9.4.2).

States: The conditions or situations tackled by the theoretical framework, offering a snapshot of the potential circumstances or statuses applicable to the framework. The Scope in Table 9.1 captures a high level view of potential states the theory encompasses. Presently applicable to individual blockchain OSS projects, expansion beyond this field is a primary extension, discussed more in Section 11.4.

Events: The dynamic dimensions of phenomena within the theory. The SEM, with its directional relationships, captures event evaluation. Should general interest be measured as defined in model SEM-4, a subsequent response to developer engagement is anticipated, illustrating the time-lag between the influence of interest on developer engagement.

Thus the theory clearly delineates the constructs, associations, states, and events which make up the individual parts.

9.4.2 Evaluation of the Whole

The evaluation criteria for a theory as a whole encompass the attributes: importance, novelty, parsimony, level, and falsifiability (Weber, 2012), offering a multi-faceted approach to assessing a theory's quality, relevance, and contribution to the field.

Importance: A theory's importance is evaluated based on the significance of its focal phenomena. While the presented research introduces a novel theoretical framework, its importance or utility within the academic and practical realm is yet to be fully established. The assessment of its significance unfolds over time as scholars and practitioners engage with the theory, apply it to real-world contexts, and potentially cite it in future works. This gradual exploration and utilisation provides a more robust and comprehensive evaluation of the theory's impact and importance in addressing pertinent issues and contributing valuable insights to the field.

Novelty: This research stands as a primary work in its domain, merging perspectives from blockchain software and insights into the discourse. While the study is novel in its approach and findings, it builds upon and integrates established literature in OSS and software health. Contributions are summarised in Table 11.1.

Parsimony: Quality theories exhibit parsimony, efficiently explaining and predicting focal phenomena with a minimal number of constructs and associations (Weber, 2012). Excessive constructs can easily emerge from EFA and SEM and associations can complicate the theory, detracting from its utility and precision. The present theory consisting of three factors and two structural relationships strives for the balance between parsimony and signal.

Level: In aligning the level of the proposed theory, it is positioned between macro and micro levels, resonating with Merton's concept of middle-range theories. These theories, as articulated by Merton (1957), avoid the extremes of dealing with the entire system or a single interaction, providing a useful framework for examining specific aspects of phenomena while maintaining a degree of generality.

Falsifiability: A theory's quality is also determined by its falsifiability. High quality theories can withstand robust empirical testing across diverse conditions, offering clear, precise predictions and suggesting potential empirical work that could lead to their falsification. This is seen in component five in Table 9.1 – Testable Propositions that offer researchers areas to strengthen, test, and refine the theory.

While the evaluation criteria for the theory are met, there are limitations which are discussed in the following section.

9.5 Limitations

The following limitations are recognised that may limit the generalisability and applicability of this theory of OSS blockchain software health. Split into three categories: general theoretical consideration, specific methodological and numerical concerns, and broader contextual factors.

9.5.1 General Theoretical Considerations

These limitations encompass the fundamental aspects of the theory itself, addressing the broader conceptual underpinnings and the potential constraints they impose on the theory's applicability and generalisability.

Scope Limitation: The theory is specifically tailored to open source blockchain projects, and the boundary statement excludes its application to business or corporate software development structures. This may limit the generalisability of the theory to other contexts or domains.

Causal Complexity: The theory posits direct causal relationships between interest, engagement, and robustness. However, real-world interactions may be more complex, involving indirect, reciprocal, or conditional relationships that are not captured in the current model.

Lack of Temporal Dynamics: The theory does not explicitly account for temporal dynamics or the evolution of constructs over time. The relationships between interest, engagement, and robustness may change as the project matures, and this aspect is not addressed in the theory.

Nascent Body of Knowledge: There is a dependency on existing literature in software health. While grounding the constructs in existing literature lends credibility, it may also constrain the theory's ability to capture novel or emerging aspects of open source blockchain software health that are not yet well represented in existing research.

9.5.2 Specific Methodological and Numerical Concerns

Two limitations are highlighted here that pertain to the specific methods, assumptions, and metrics used in the theory. Further, more detailed limitations in the methodological decisions, data collection, and analysis are in Chapter 4.

Measurement Challenges: The latent factors of interest, engagement, and robustness are derived from specific metrics (for example, forks, stars, mentions, criticality). While these metrics are empirically grounded, they may not capture all the nuances or dimensions of the constructs, leading to potential measurement bias.

Assumption of Linearity: The structural equation model implies a linear relationship between constructs. However, non-linear relationships, threshold effects, or saturation conditions may exist in some cases, and these are not represented in the theory.

9.5.3 Broader Contextual Factors

This category recognises the limitations related to external influences and qualitative aspects that are not explicitly addressed within the theory. These include broader societal trends and factors that may affect the constructs but are not within scope of the current theory.

External Factors: The theory focuses on internal dynamics within the open source community but may overlook external factors such as market trends, regulations, or technological advancements that could influence the constructs.

Potential Overemphasis on Quantitative Metrics: The reliance on specific quantitative metrics may overlook qualitative aspects such as community sentiment, developer motivation, or organisational culture, which could provide additional insights into the health of open source projects. These limitations highlight areas for caution, refinement, or further exploration. They also provide avenues for future research.

It's essential to recognise that theories, particularly in the realm of sociotechnical systems like OSS blockchain software health, serve as anchoring points for understanding and investigation rather than definitive encapsulations of complex reality. The inherent complexity and multifaceted nature of sociotechnical systems mean that a theory, while valuable for structuring insights and guiding inquiry, cannot capture all the nuanced behaviour and dynamics at play. The limitations of this theory are not merely constraints but opportunities to deepen understanding, challenge assumptions, and innovate methodologies. They invite a continual process of inquiry, adaptation, and growth that acknowledges the fluidity and richness of the subject matter (Weick, 1995).

Theoretical frameworks function as a foundation, but they must be flexible and responsive to the evolving landscape of technology, community, market forces, and other variables that interact in intricate and often unpredictable ways. The pursuit of comprehensiveness must be balanced with the recognition that many factors are ultimately outside comprehension and ability to measure and judge them. The theory's value lies not only in its explanatory and predictive power but also in its capacity to inspire curiosity, critical thinking, and ongoing engagement with the complex, dynamic nature of OSS and blockchain technology.

9.6 Conclusion

This chapter articulates the development of a theoretical framework focused on the health of OSS blockchain projects. Through a systematic examination of the constructs of interest, engagement, and robustness, the framework offers a structured approach to understanding the complex dynamics within this domain.

While incorporating elements often associated with theory development, such as causal relationships and testable propositions, it is essential to recognise that this work represents a theoretical framework, or perhaps a framework that leans in a theoretical direction, rather than a fully-fledged theory. To expand on the quote in the Chapter's epigraph (Page 179):

> The process of theorizing consists of activities like abstracting, generalizing, relating, selecting, explaining, synthesizing, and idealizing. These ongoing activities intermittently spin out reference lists, data, lists of variables, diagrams, and lists of hypotheses. Those emergent products summarize progress, give direction, and serve as placemarkers. They have vestiges of theory but are not themselves theories. (Weick, 1995, p.6)

The framework provides a foundation, delineating key concepts and relationships, but it does not posit a complete theoretical structure.

The contributions of this framework are multifaceted, offering new insights into the constructs of interest, engagement, and robustness, and their interplay within the context of open source blockchain projects. However, limitations include challenges related to scope, measurement, and potential biases.

As a tool for researchers and practitioners, the framework lays the groundwork for further exploration and analysis. The implications extend to the broader field of software health, offering potential insights for various contexts within the open source community.

The development of this theoretical framework represents a step in the ongoing investigation of open source blockchain project health. It serves as a foundation upon which future work may build, potentially evolving into a complete theory through empirical validation, refinement, and extension. In this regard, the framework contributes to the evolution of research within the field, offering a structured lens for inquiry and a platform for further academic dialogue and innovation.

Chapter 10

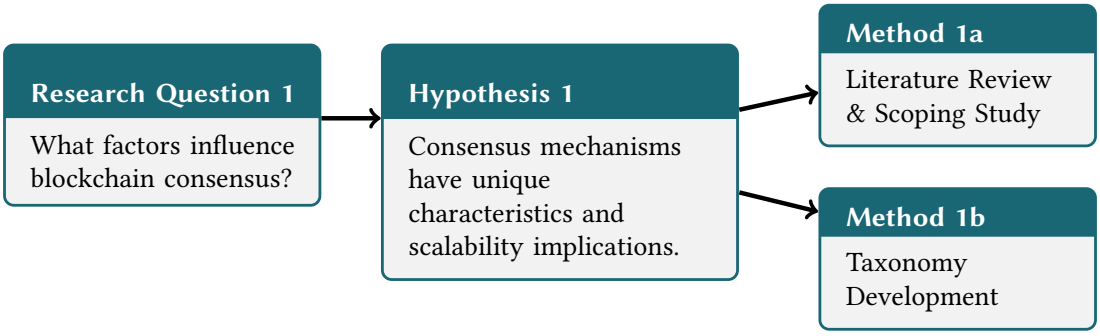
Discussion

> *The whole of science is nothing more than a refinement of everyday thinking*
>
> Albert Einstein
> *Physics and Reality*
> 1936

The Penultimate Chapter synthesises and interprets the research findings. First, the research objective is reviewed. Subsequently the findings are discussed in the context of each research question originally detailed in Section 4.1 and seen in Figure 4.3. Here, limitations are addressed in the context of the hypotheses and methods. Finally, the contextualisation within the field of Information Systems is discussed.

10.1 Blockchain Consensus and Scaling

The first research question asking what factors influence blockchain consensus and scaling leads to the development of the taxonomy presented in Chapter 5.



The work begins with an investigation into blockchain consensus methods. The impetus for this line of inquiry stems from a widely acknowledged limitation in blockchain systems—their restricted capability for scalability in terms of user base and data throughput. The study finds scalability is intrinsically tied to the employed consensus algorithms.

Hypothesis 1 posits that distinct consensus mechanisms exhibit unique characteristics that have consequential implications for system scalability. To evaluate this assertion, two methodological approaches are deployed. Initially, a literature review and scoping study are undertaken to identify and assess the existing body of knowledge concerning blockchain consensus mechanisms. Subsequently, a taxonomy is formulated in Chapter 5 to systematically classify the inherent characteristics and scalability ramifications of these consensus mechanisms within the blockchain domain.

The taxonomy in Table 5.3 is subjected to a peer review process and is additionally validated by its applicability to consensus algorithms that fall outside the predefined boundary conditions delineated in the initial literature review (Table 5.4). The taxonomy thus exhibits both extensibility and generalisability. It accommodates not only extant consensus mechanisms but also demonstrates the potential to incorporate novel methodologies with regards to future research.

Furthermore, the taxonomy serves a dual purpose. The landscape as presented in the taxonomy not only shows what exists from the literature, but also can show what does not exist and perhaps could be areas for future research. These areas for future research lead to innovation in blockchain software.

Subsequently, the analysis pivots to focus on the broader ecology of OSS development within the blockchain sphere. The underlying premise is that for OSS projects to serve as incubators for innovation, they must be situated within an ecosystem that is conducive to software development, a more generalised concept of 'software health'. The remainder of the research is focussed on determining, elucidating, and modelling software health in the blockchain ecosystem.[1]

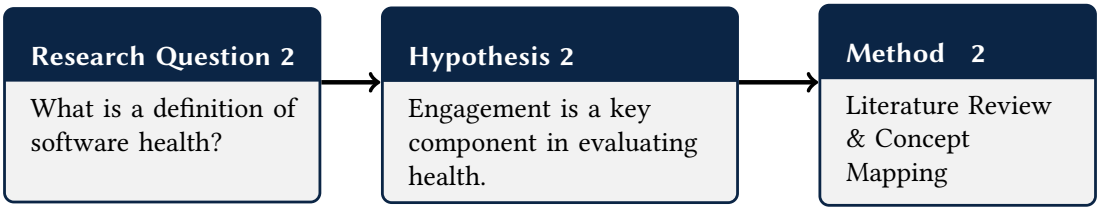
RQ1 Summary

In summary, to answer RQ1: What factors impact blockchain consensus and scaling? Consensus methods and scalability are strongly linked in the literature; the taxonomy in Table 5.3 categorised consensus methods according to their unique attributes which supports H1.

10.2 Definition of Software Health

Having delineated the role of specific consensus mechanisms in blockchain scalability, the inquiry is broadened to include the overall well-being of OSS projects within the blockchain ecosystem. This transition is not merely tangential but underscores the interconnectedness of the various components that contribute to the success and scalability of blockchain technologies. At the heart of this is software health, a term without broad agreement on a definition.

The second research question asks what is a definition of software health and leads to the definition of health presented in Section 3.2.2.



The conceptual mapping in Table 3.1 finds that ecosystem health is made up of three factors: sustainability, robustness, and niche fit. The research narrows this down to the level of

1. Blockchain scaling is briefly revisited at the end of Section 11.4.

individual projects, as depicted in Figure 6.1. At this granular level—specifically within the realm of GitHub repositories—software health is defined as a confluence of three elements: engagement, interest, and robustness. As per the figure, engagement and interest collectively constitute sustainability. The closest definition is from (van den Berk et al., 2010) for software ecosystems which is similar to (Iansiti & Levien, 2004) for business ecosystems. Both authors use productivity, robustness, and niche creation to define health. The present work replaces productivity with sustainability and further differentiates between the idea of engagement and general interest. The nearest definition of health within OSS comes from (Goggins et al., 2021) consisting of sustainability and survivability but does not consider the niche position in the broader ecosystem. Of all the version of health described in Table 3.1, there are no broad deviations from one another, it is conceivable to believe any definition in isolation. Rather, they present the limitations of attempting to demonstrate and operationalise an illusive concept such as health.

Hypothesis 2 for this research question states that engagement serves as a key metric for assessing the health of a software project. Testing of the hypothesis is by two primary methodological approaches. First, a comprehensive review of the existing academic and industry literature on the metrics and indicators used to evaluate software health. Second, a conceptual mapping to identify and group the factors that contribute to software health.

Testing the hypothesised claim that engagement is a key factor in health evaluates to be true. In support of H2, Table 3.2 shows that of 25 metrics related to software sustainability, 18 of them are classified as engagement metrics and the remaining 7 as general interest metrics. Additionally, according to the relevance in the literature, the top two metrics that appear across six sources each are bug-fix rate, and comments. Both of these metrics are engagement-type metrics, although bug-fix rate does not emerge in the present study (see Section 10.3.1). The next most broad metric is contributor count which appears in five sources, which is also an engagement metric.

H2 is further validated statistically by the EFA in Figure 6.9 which determines a latent factor, now called developer engagement, from the set of possible metrics identified in the literature. CFA (Table 8.4) is also in agreement which places engagement as a latent factor whose composite metrics account for 73.1% of variance extracted from the data in the model representation.

RQ2 Summary

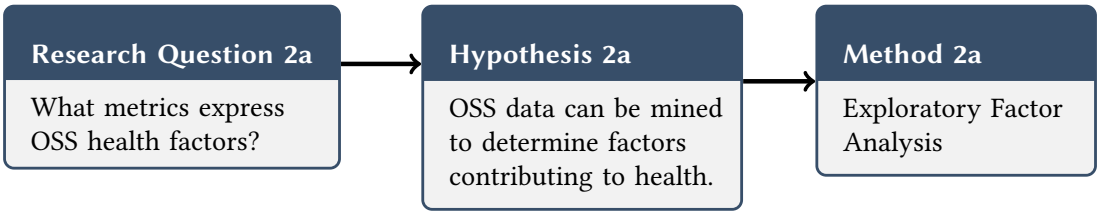
In summary, to answer the RQ: What is a high-level definition of software health? A definition is determined from a conceptual mapping of the literature (Section 3.2.2).

The related hypothesis H2: Engagement is a key component in evaluating health is supported.

10.3 Health Metrics

Having established the conceptual underpinnings of OSS health, the focus shifts to the quantifiable aspects of this health. RQ2a in addresses this by posing the question: What health metrics can be identified? This inquiry not only refines the understanding of software health but also allows for empirical evaluation, thereby adding robustness to the overarching investigation.

Hypothesis 2a posits that data from OSS projects can be effectively mined to ascertain factors contributing to health. To examine the feasibility and implications of this proposition, EFA is employed as Method 2a. The EFA aims to uncover the latent variables that can be considered as measurable indicators of software health, thereby providing a statistical basis to the qualitative discussion of RQ2.



The EFA is conducted in two parts. First, indicator metrics for developer engagement are found in Chapter 6. Results of the EFA show that four indicator variables compose a latent factor: pull requests, comments, authors, and commits, which is subsequently named to represent the construct of developer engagement. Figure 6.9 shows the resultant factor loadings after the EFA process. PRs has the strongest influence on engagement, followed by comments, authors, and commits. When considering the measurement model inclusive of all indicator variables, comments moves to the strongest influence (Figure 8.4). Both comments and PRs are the most influential metrics across the studies. This is not surprising as they require knowledge of the context and codebase. What is of note is that raw number of commits[2] has the least

2. Commits is calculated as a monthly average over the previous three months.

influence of the group of indicators on developer engagement. Thus less emphasis should be placed on commit numbers when evaluating engagement, and more weight is applied to the comments participants are making.

In the second part, indicator metrics for general interest and robustness are found in Chapter 7. Software robustness consists of criticality score, time since last update, CoinMarketCap rank, and geographic distribution. The last latent factor is composed of forks, stars, and mentions and is named general interest. Figure 7.7 shows the factor loadings after the EFA process. The influence from strongest to weakest on robustness is: criticality score, time since last update, CMC rank, and geographic distribution. Forks and stars load strongly on interest with mentions a close third in influence. Both of these hierarchies hold when considering the measurement model (Figure 8.4). Software robustness is the weakest latent construct with two indicators, CMC Rank (0.391), and geographic distribution (0.375), loading near EFA/SEM thresholds which are between 0.3–0.4. This highlights the difficulty with defining the robustness construct and operationalising indicator metrics. Another item to note is that criticality score loads highly on robustness (0.998) which can outweigh the other indicators. Also, criticality score is a composite measure, made up of multiple components, and its relationship with the latent variable might not be straightforward. This complexity can make it challenging to disentangle the effects of its individual components on the Robustness. Stars and forks, as part of the general interest construct, are discussed below.

10.3.1 Discussion on the Identified Metrics

The metrics that emerge from the EFA do not necessarily mirror the literature on software health both in regards to exclusion (time to resolve issues) and inclusion (general interest of stars and forks) and overall absence of niche occupation.

What happened to bug-fix time?

Despite the prevalence of bug-fix rate as a key metric in the sustainability and engagement dimensions of software health (Table 3.2), the current dataset yields no evidence to support its individual importance. In line with existing literature, the present study initially incorporates two metrics to gauge this rate: the median and average times between an issue being opened and subsequently closed. Both measures are included in the EFA to ascertain whether their different statistical properties would yield divergent insights. They are not, however, included in the analysis for developer engagement. Rather, for a system to be robust it must adapt to issues that arise, especially critical ones that can result in downtime and lost functionality and customers; for this reason, bug fix time is included in the indicator variables for the factor

representing robustness (Section 7.3).

However, neither the median nor the average time for bug-fixing is integrated into a structural relationship within the resulting model. These two indicators are originally treated as separate, independent factors, and eliminated from the model individually in Section 7.7.1. Consequently, any calculated measurement of issue resolution rate is omitted from the final model.

This omission does not negate the importance of bug-fixing in software health. Bug-fixing is indeed a second-order result of baseline developer activity and thus subsumed under the engagement latent factor, as supported by (Z. Wang et al., 2020). Contributors often commence their involvement in a new project by addressing open issues, which results in bug-fixing activities. These activities are encapsulated within the engagement latent factor through the first-order indicators such as commits, comments, and pull requests. Additionally, omitting a direct measure of bug-fix rate contributes to model parsimony by reducing indicator redundancy.

`Where does general interest or popularity fit into software health?`

Jansen (2014) categorises interest as part of robustness whereas the present study finds interest to be a part of sustainability and finds it has no direct influence on robustness. The *Interest* → *Robustness* loading is insignificant at −0.06 (Figure 8.9).

This raises questions about the impact of popularity metrics on software health. While it seems interest can contribute to the robustness of a project by increasing its popularity and attracting more developers, it is more likely that increased popularity increases engagement which then affects robustness. One of the benefits of SEM is being able to disambiguate this relationship.

Forks and stars are strong interest metrics, with both being used by Osman and Baysal (2021) to define popularity, and by Abdulhassan Alshomali (2018) to define repository interest. Additionally, Negoita et al. (2019) have stars as a sole definition of sustainability. As Jansen points out, once a competitor emerges, users may shift their attention to a more promising alternative, potentially causing long-term damage to the original project. In this sense, the concept of a popular project may align more closely with sustainability, rather than robustness.

Stars and Forks

In Section 6.8 it is mentioned that stars and forks are independent metrics both to be included in the model. The EFA is conducted with both included, with stars included and forks excluded, and vice versa and it is decided both metrics remain included. The high correlation between stars and forks (0.98, Figure 7.3) would suggest that they measure the same thing and yield an over emphasis in the model results which is seen by the factor dominance in ML_5 in Table 7.6.

The first SEM, denoted as CFA-1 in Table 8.2, further illuminates this issue. In this model, stars and forks are uncorrelated, yet forks exhibited an outsized influence, even yielding a slight negative variance (−0.002). Subsequent models from Section 8.5.2 subsequently corrected this by introducing a correlation between stars and forks.

However, the Bayesian network path analysis, discussed in Section 10.4, challenges this correlated understanding of the two metrics. Figure 8.14 designates forks as the root node and stars as a leaf node, thereby inferring that forking is a more consequential activity in relation to the generation of robust software than is starring. This calls into question not just the high correlation between stars and forks but also the extent to which stars contribute to the latent factor of general interest. In its absence, the latent factor for interest could be left underdetermined, represented only by forks and mentions. Consequently, further investigation is required to identify metrics that are truly representative of the latent construct of interest.

Niche Fit Metrics

From the baseline definition of software health, that has a parallel in ecosystem health (Section 3.2.1), the local context of a species or project in the ecosystem is deemed important. This niche occupancy has sound logic: if a software project fills a specific market gap and has no competition it is positioned to thrive, and, more likely to be healthy. The metrics in Table 3.4 present the complex issue of how exactly to identify the niche and quantify it. The perspective here is that of the individual software project which limits the context required to determine if it fits a niche (is unique) or not (has strong competitive alternatives). In other words, this is not analysing the local environment to see if what the project delivers fills a niche, rather the empirical approach is content agnostic, and seeks to determine health without the subjective approach of determining market fit, or other such niche indicators. As such, there is limited research to operationalise niche metrics in OSS.

Chengalur-Smith et al. (2010) define the construct of niche through audience niche, programming language niche, and operating system niche. What audience niche means is unclear, however language and OS niche are if the project has support for less popular languages and platforms. The study finds that none of these metrics have a significant effect on attrac-

tion or sustainability, with niche size path estimates of less than 0.04. While this suggests that more research is needed to determine the relationship between niche occupation and software health, it also highlights the complexity of measuring and interpreting these metrics in the context of collaborative software engineering.

Although the niche occupancy is defined as out of scope in Section 6.3.5 this area of inquiry presents as a limitation to the current study, and an avenue of work for future studies.

10.3.2 General Limitations with EFA

It must be noted that "factor analysis will always produce factors" and is agnostic to "garbage in, garbage out" (Hair Jr. et al., 2014, p.97). Therefore, the role of the researcher is to ensure appropriate indicators are chosen, data is carefully collected, and factor results are thoughtfully interpreted.

Beginning with data collection, a few threats are notable. First, the data is only sourced from GitHub. This is due to the prominence of blockchain projects being hosted here, but must be considered. Secondly, repository owners can move and rename repos in the time between manual verification of the location and the database query time. Although this renaming is rare because it breaks any external links to the codebase, in this instance they may be missed or counted twice.

Concerning sample sizes (Section 6.3.2), the study *Minimum Sample Size Recommendations for Conducting Factor Analyses* by Mundfrom et al. (2005) suggests that, for optimal criterion with three-indicator factors in a three-factor analysis, up to 600 data points are necessary. The current research approaches this threshold, yet achieving it is challenging due to the nascent state of the industry, which limits the availability of mature software projects for inclusion. The initial dataset comprised the top 600 projects, albeit with the understanding that private repositories would not be accessible for analysis. The theoretical possibility of extending the dataset to the top 1200 projects exists; however, this approach may be subject to diminishing returns, as it risks incorporating more inactive projects that may not be representative of the intended sample. Additional constraints include the time commitment required for manual verification of each project prior to data collection, given that automated scripts proved insufficient for error detection in code location. One avenue for future work could involve expanding the dataset to encompass additional repositories within an organisation.

On fit statistics used to verify the models (Section 6.6.4), the thresholds commonly recommended are developed in the context of normally distributed data (Finch, 2020b), and must not be considered law. Acceptance or rejection of models should not be based on fit statistics, rather on the ability of the model to provide structure to the data. In this spirit, statistics should be considered, published (as is the convention), and used to provide broad support and

directional confirmation rather than a target threshold to meet.

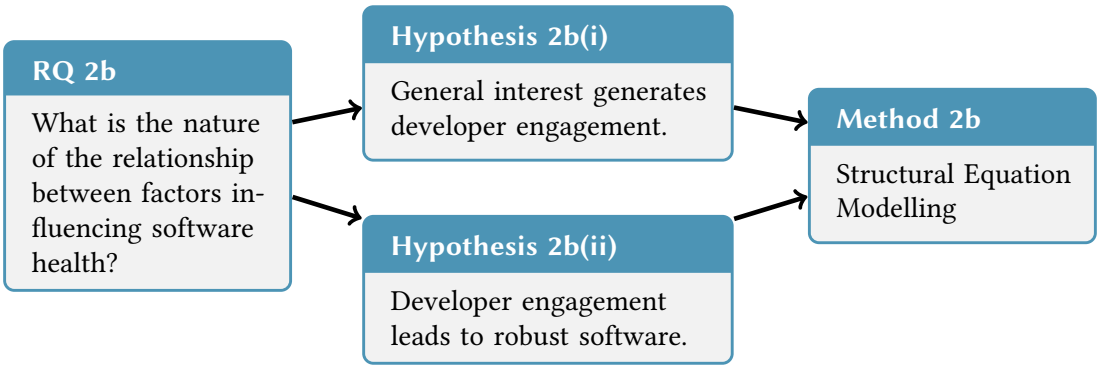
H2a Health Metrics Summary

In summary, to answer RQ2a: What health metrics can be identified? The literature review identifies a pool of metrics (Tables 3.2 to 3.4) and the EFA procedure identifies operationalised sets grouped into latent factors in Chapters 6 and 7.

The hypothesis: OSS data can be mined to determine factors contributing to health is supported under the definition of software health.

10.4 Structural Relationship

Following the identification of metrics that delineate software health, attention now turns to understanding the structural relationships among factors. RQ2b seeks to investigate the relationship between general interest, developer engagement, and software robustness. Two hypotheses are formed:



SEM serves as the methodological approach here. SEM enables a simultaneous analysis of multiple relationships, complemented by the validation of the model through fit statistics and diagnostic measures.

10.4.1 Is Healthy Software Robust Software?

Goggins et al. (2021) agree with Chengalur-Smith et al. (2010) in their definitions of health as a combination of sustainability and survivability as shown in Table 3.1. The concept mapping uses the term robustness rather than survivability, and if these are treated as synonyms for a moment the logic can be illustrated. A project can be sustainable but not survive. However, a project cannot survive without being sustainable. Surviving projects must therefore also be

sustainable. This is supported by the path relationship *Engagement* (part of sustainability) → *Robustness*, in other words, survivability (Figure 8.12).

The study sought to model health as an endogenous latent factor[3] within the SEM framework. However, no significant results are obtained due to the absence of indicator variables explicitly contributing to health. This shortcoming can be attributed to two factors.

Firstly, health is intrinsically a nuanced concept with a subjective nature. As seen in the literature review and further elaborated in Section 3.2.2, health is a second-order construct. It is a state of a system that necessitates an understanding through composite latent variables. Determining these composite latent factors is a goal of this study, thus contributing to a picture of health, but not attempting to define health in its entirety.

Secondly, demarcating health as an endogenous latent factor would be inherently limiting, as it would confine its defining variables to those identified within the scope of this research. The multifaceted nature of health, possibly involving complex social dynamics not examined here, could then be inappropriately constrained. This limitation also has implications for the potential generalisability of the theoretical framework presented.

An initial structural model is posited that conceptually linked health to interest, engagement, and robustness. However, this model proved numerically unsolvable using `lavaan` with the present data, leading to the abandonment of this line of inquiry.

10.4.2 Discussion on SEM Validity

Validation of a SEM is primarily by statistical fit of the model simulation to see how well the hypothesised structural model fits the observed data. This is completed in Section 8.7. This is in concert with the researcher's expertise to gauge the nomological validity of the model. Nomological validity ensures that the observed relationships are not merely statistical artefacts but reflect underlying theoretical frameworks (discussed next, in Section 10.5). If the SEM results are not acceptable, then the researcher returns to the model definition. If the SEM is acceptable, progress can be made on the theoretical implications.

Further validation is by two methods. First, in Section 8.9.1 the model is bootstrapped by simulating the sampling distribution through resampling, to provide robustness checks for parameter estimates, enhancing the reliability of conclusions drawn from the model. Second, the model is evaluated via a BNPA in Section 8.9.2.

3. An exogenous latent factor is composed of observed variables, whereas an endogenous factor depends on exogenous factors.

10.4.3 Bayesian Network Path Analysis

BNPA, discussed in detail in Section 8.9.2, serves as a secondary validation mechanism for the SEM. The Bayesian DAG, illustrated in Figure 8.14, is constructed from the dataset without any pre-defined whitelisting or blacklisting constraints, employing a hill climb optimisation algorithm.

Observing the colour-coded grouping of indicator variables, which correspond to latent factors, reveals a natural hierarchical clustering. Each directed edge in the DAG represents an influence relationship, and, when grouped according to latent factors, all but two paths proceed from root to leaf nodes in the direction modelled by the SEM, as seen in Figure 8.13.

Two divergent pathways, specifically comments to stars and authors to stars, challenge conventional wisdom regarding the primary importance of stars as a measure of general interest. These pathways indicate that while comments and authors might lead to an increase in stars, the metric of stars has limited downstream influence on other variables. Stars, being a leaf node, serves as an end result rather than a causal factor for other variables.

Moreover, the other leaf nodes in the DAG—specifically, CoinMarketCap ranking, geographic distribution, and last updated—fall under the robustness latent factor. This reveals that robustness is consequential to other activities, akin to a dependent variable, rather than an isolated factor. Consequently, any endeavours to enhance robustness would necessitate prior improvements in areas of engagement and interest.

The BNPA thus not only reinforces the SEM but also highlights complexities and nuances that are less evident in the SEM, particularly with regard to the influence hierarchy among the variables.

10.4.4 General Limitations with SEM

One of the primary limitations of SEM in this study is its reliance on the same dataset used for the EFA. Consequently, the quality of the statistical results derived from the SEM is bounded by the limitations inherent in the CFA. Statistically, SEM does not offer improvements in the rigour of the results over CFA. However, it does offer an additional layer of explanatory power by assigning constructs and explanation to data that might otherwise be interpreted intuitively.

Model misspecification represents another significant limitation inherent to SEM as a methodology. Such misspecification can manifest in various forms: an overly complex model, insufficient data to justify the parameters, or both. Advanced techniques like modification indices can be used to improve the fit statistics of the models (also for CFA), but must be judiciously applied. These model respecification parameters are investigated, but not considered

seriously as they lack grounding. The model under discussion, as presented in Figure 8.12, is designed with parsimony in mind, featuring only two structural pathways, thereby avoiding the criticism that complexity has obfuscated clarity.

As for the data, a natural extension for future work (Section 11.4) is to collect a new dataset to test the model's validity. Unfortunately this is out of scope of the current work.

10.4.5 Statistics for Statistics Sake

Lastly, a note on statistical methods. Due to the intricate statistical nature of SEM, there exists a risk that researchers may deploy the method indiscriminately to discern factors and relationships. In such cases, the ensuing models could exhibit apparent statistical rigour while lacking substantive theoretical or practical interpretability. This methodological overextension may result in findings that are statistically plausible but conceptually spurious, thereby undermining the credibility and applicability of the research.

The validity of the SEM depends on the CFA, which in turn depends on the EFA. The factor derivations and data collection all depend on the definition of software health from the literature review. This shows a solid scaffolding of result validity upon which the theoretical framework (Chapter 9) is derived from.

In summary, given the background in Chapters 2 and 3, methodology in Chapter 4, and foundational elements in Chapters 6 to 8 that are represented in the present work, RQ2b is answered and hypotheses supported.

H2b(i & ii) Structural Relationship Summary

In summary, RQ2b asked: What is the structure between the software health components? This is answered through the SEM process resulting in model SEM-4 in Figure 8.12.

The hypothesis: Public interest generates developer engagement is supported.

The hypothesis: Positive developer engagement leads to robust software is supported.

10.5 Theoretical Framework

Transitioning from the relationships among software health components in RQ2b, the natural progression leads to RQ3: What is included in a comprehensive model of blockchain software

health? Herein lies the synthesis of all preceding conceptual and empirical findings into a cohesive framework. RQ3 aims to articulate a comprehensive model that not only integrates individual metrics and their interrelations but also provides actionable insights to inform innovation in the domain of blockchain OSS.

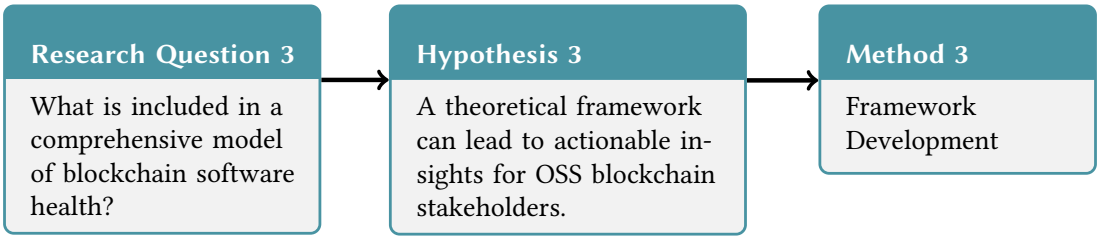


Figure 9.2 shows the primary theory representation of the framework for OSS blockchain health. The style of the theory is the explanation and prediction type.

10.5.1 Explanation and Prediction

Both explanation and prediction serve as dual facets of theory that engage deeply with causal relationships. For instance, uncovering causal pathways can not only clarify the nature of observed relationships but also facilitate predictive modelling of future occurrences.

These causal insights are instrumental in shedding light on the underlying mechanisms and processes that engender the observed relationships, thus fulfilling the framework's objective of providing comprehensive explanations. While the treatment of causality should be approached judiciously, its relevance in deepening the understanding of phenomena should not be sidelined.

The fifth element of the theoretical framework in Section 9.3.5 is causal explanations. These explanations serve as critical elements, shedding light on the mechanisms and processes that underpin observed relationships. The SEM elucidates two pivotal causal explanations: firstly, that general interest is a precursor to developer engagement, and secondly, that heightened developer engagement enhances software robustness.

This causal structure allows for direct practical statements to be made, whether in the form of hypotheses for future investigation or advice for practitioners. Recalling the goal of the theoretical framework from Section 9.1, it is to help practitioners and researchers answer the following questions:

1. How can an open source project be structured to ensure optimal health?
2. How to evaluate current projects for health status, and provide guidance for improvement in software health?

Based on the causal structures delineated through the SEM, fostering public interest is crucial for subsequent developer engagement. Thus, initial project setup could involve public awareness campaigns, open documentation, and an initial codebase that aligns with community interests to attract a broad spectrum of developers. Transparent governance structures and clear contribution guidelines could also be instituted to facilitate sustained developer engagement, which, in turn, leads to software robustness.

Evaluating the health of an existing project could be conducted through an audit using metrics corresponding to the latent factors identified in the study (for example, forks, stars, mentions for interest and engagement, criticality measures for robustness). If the project lacks in any aspect, targeted interventions can be implemented. For example, if developer engagement is low despite high public interest, one could investigate communication channels, review processes, or other barriers to participation.

The previous two examples are not meant to be definitive directions, nor is this considered a result of the present work. The framework as an academic artefact is meant to provide an empirically-grounded methodology for the assessment and interpretation of health indicators in open source projects, facilitating targeted interventions where necessary. Additionally, a conceptual lens through which researchers can examine the interplay between different facets of software health, thereby opening avenues for future inquiry.

10.5.2 Framework Application

The framework presents opportunity for further research by uncovering hypotheses. A few examples are shown here. The following 2nd layer hypotheses provide deeper insights and more nuanced directions for researchers who may want to investigate further into the dynamics of interest, engagement, and robustness in OSS blockchain health.

Hypothesis 1a: Different types of interest (for example, academic, commercial, hobbyist) have varying degrees of impact on developer engagement.

Investigating the nuances of how specific forms of interest influence engagement may reveal targeted strategies for fostering community collaboration.

Hypothesis 1b: The effect of interest on engagement is moderated by the maturity stage of the project.

Early-stage projects might be more sensitive to fluctuations in community interest, while mature projects may exhibit more stable engagement patterns.

Hypothesis 2a: Engagement's positive impact on robustness is mediated by specific practices, such as coding standards, peer review, or collaborative problem-solving.

Understanding these mediating factors can help in designing effective development processes.

Hypothesis 2b: The relationship between engagement and robustness is contingent on external factors, for example, technological trends, market competition, or regulatory environment.

Exploring these contingencies can offer insights into how external pressures shape software robustness.

Hypothesis 3a: The collective reflection of health through strong robustness involves a feedback loop where improved health attracts more interest, leading to a virtuous cycle.

Examining this feedback mechanism may uncover dynamics that sustain long-term project vitality.

Hypothesis 3b: The alignment of robustness with a state of health varies across different types of open source blockchain projects, for example, infrastructure projects, application layer projects, or experimental initiatives.

Comparative analysis across these types may reveal unique pathways to health within the diverse landscape of blockchain development.

As a tangible artefact in Design Science, the framework provides opportunity to advance knowledge in addition to the goals of explaining and predicting. By probing these second-layer questions, researchers can build a richer and more clear understanding of the factors that contribute to the health of open source blockchain projects.

10.5.3 Framework Limitations

The limitations are address in Section 9.5 and briefly reiterated here. As the framework is tailored to OSS blockchain projects, the theory may not be universally applicable. It assumes direct causal relationships among variables but may not capture complex, indirect, or evolving relationships. Temporal dynamics are overlooked, and the model relies on specific, empirically-grounded metrics, introducing the risk of measurement bias. The assumption of linearity in the SEM is another constraint, as is the neglect of broader contextual factors such as market trends and qualitative community aspects.

Despite these limitations, the theory offers a foundation for further research and adaptation in the landscape of blockchain technology and OSS health.

RQ3 Theoretical Framework Summary

In summary, the RQ: What is included in a comprehensive model of blockchain software health? is answered by the theoretical framework presented in Figure 9.2 as described in Chapter 9.

10.6 Contextualisation

Contextualisation places the results of the present study within the broader intellectual landscape of IS, offering a contrast between the framework's contributions and existing theories, models, or empirical findings. By doing so, this section aims to underline the significance of the research outcomes, outlining how they corroborate, challenge, or extend the current body of knowledge.

Alignment with Existing Theories

The framework engages with the existing IS literature, particularly aligning with and expanding upon the DeLone and McLean (1992) model of Success in Information Systems, as referenced in Section 3.2.3. Although *success* and *health* are often used interchangeably in the literature pertaining to software systems, this work adopts a nuanced approach, choosing to focus specifically on the constituents of health rather than providing predictive claims about success.

Figure 3.5 elucidates a model of Health as IS Success. This adaptation draws on a literature review conducted on the topic of ecosystem health, thereby situating the research in a broader theoretical context. Importantly, while the adapted DeLone and McLean model serves as a structural foundation for the theoretical framework developed for OSS health, it is noted that the model has yet to be subjected to evaluation by Gregor's criteria for theory in IS or by Weber's evaluation of theory in IS. In this sense, Figure 3.5 functions more as a signpost along the path to the theoretical framework presented in Chapter 9.

Moving towards theories that have software as a focus, Crowston et al. (2006) adapt DeLone and McLean's model to OSS. They overlay software to the components: inputs as the number of developers, process as the time for bug fixes, and outputs as the popularity which is measured by downloads, users, views, and republishing. This success perspective is different from the health perspective, although both employ metrics such as developer count and popularity (stars). The success perspective is seeking to collate completed software projects with successful software projects. The health perspective is more interested in the state of the software project with the aim that it will continue to survive. An interesting view is that

general interest is on the input side of the framework in Figure 9.2 and the output side of Crowston et al.'s model. Similarly, developer count is on the input side of Crowston et al.'s model whereas its an endogenous construct in the SEM modelling. The success models are not elucidating the concept of being robust to environmental shocks, and so here a health model is notably different.

10.7 Conclusion

To conclude, the research questions and hypothesis are each discussed, beginning with a summary of the method and results, then turning to the validation and limitations. In Chapter 11 the research contributions are summarised and some of the future research directions are pointed out.

Chapter 11

Conclusion

> *In conclusion, there is no conclusion. Things will go on as they always have, getting weirder all the time.*
>
> Robert Anton Wilson

This Concluding Chapter begins with a succinct reiteration of the research objectives. Subsequently, the contributions to the existing body of knowledge within the field are enumerated. Thereafter, the practical implications of the study are listed. Finally, avenues for future research are outlined, as it is often the case that rigorous inquiry reveals more questions than it answers, thus beckoning further, and weirder, exploration.

11.1 Research Objectives

This research is motivated by multidimensional complexities of OSS health, and blockchain scalability and consensus. Motivated by the hypothesis that an enhancement in blockchain performance is not just a matter of technological improvement but also significantly tethered to the health of the OSS project, this study pursues the goal of determining health through the specific aims:

- To articulate a taxonomy that situates blockchain consensus within the multifaceted scaling considerations of the blockchain trilemma.
- To define the concept of health in the specialised context of blockchain-based open source software. This conceptual delineation serves as a foundation for empirical investigations.
- To perform a detailed analysis of various health factors in OSS environments.
- To construct a theoretical framework capable of assessing health of blockchain projects in the OSS domain.

These objectives are distilled into actionable research questions summarised below in Table 11.1 and are explored through methodologies as elaborated in Chapter 4. The translation of these objectives into concrete research actions forms the scaffold upon which this thesis is constructed, and it contributes to the scholarly discourse on the intricacies of blockchain and OSS health.

11.2 Contribution to Knowledge

The proposed theoretical framework on the health of open source blockchain projects, focusing on the interplay between interest, engagement, and robustness, contributes to knowledge in several significant ways enumerated here and summarised in Table 11.1.

TABLE 11.1: A summary of the contribution to knowledge showing the research questions and corresponding artefacts, cross referenced to their location in the thesis.

Research Question	Artefact	Reference
1. What factors influence blockchain consensus?	Taxonomy	Table 5.3
2. What is a definition of software health?	Definition	Figure 3.6
2a. What metrics express OSS health factors?	Measurement Model CFA-2	Figure 8.5
2b. What is the nature of the relationship between factors influencing software health?	Structural Model SEM-4	Figure 8.12
3. What is included in a comprehensive model of blockchain software health?	Theoretical Framework	Figure 9.2

The prime contribution of this thesis and result of the prior work and statistical analysis is a theoretical framework that includes the following specific contributions.

Conceptual Framework: By introducing latent factors and causal relationships, the theory presents a novel conceptual framework that extends the understanding of open source software health, particularly in the field of blockchain software.

Empirical Grounding: With the foundation in EFA and specific metrics identified, the theory provides an empirically grounded approach. This methodological contribution enhances the rigour and reproducibility of research in this area.

Integration of Multidimensional Constructs: The theory integrates multidimensional constructs that capture complex dynamics within open source blockchain projects. This integration offers a more nuanced understanding of how interest, engagement, and robustness interact and contribute to overall software health.

Actionable Insights for Practitioners: By translating theoretical constructs into prescriptive statements, the theory bridges the gap between academic research and practical application. It offers actionable insights and guidelines that can be implemented by practitioners, developers, and policymakers.

Facilitation of Future Research: The theory lays the groundwork for further investigation, including the exploration of second-layer hypotheses. It opens avenues for comparative studies, longitudinal analyses, and the examination of moderating and mediating factors, enriching the research landscape.

Enhancing the OSS Blockchain Ecosystem: Specifically, within the field of blockchain, the theory contributes to shaping best practices, encouraging community participation, and fostering developer engagement for robust software development. It directly addresses a critical area that is vital for the sustainability and success of open source blockchain initiatives.

In summary, the theory contributes to both the academic and practical realms, offering a comprehensive, empirically grounded, and actionable framework that enriches the understanding of OSS blockchain software health. The framework's focus on specific metrics and causal relationships offers a foundation for future empirical study and practical guidance. It provides a structure that facilitates explanation, understanding, analysis, and exploration of the factors influencing open source blockchain software health.

11.3 Practical Implications

The goal is to solidify research into OSS health in a manner that can provide a clear definition of, and metrics to assess, software health. As Goggins et al. (2021) say, "There is a considerable amount of research constructing and presenting indicators of open source project activity, but a lack of consensus about how indicators derived from trace-data might be used to represent a coherent view of open source project health and sustainability." This study and methodological approach allows for a reproducible empirical process for data collection, metric calculation, and indicator variable selection.

Additionally, by the definition of latent constructs, the study results allow stakeholders such as future and current OSS contributors, researchers, and project managers to identify areas for improvement in their software projects. By understanding the factors that contribute to software health, project managers can make informed decisions about where to allocate resources to improve software based on operationalised metrics attributed to interest, engagement, and robustness. The study provides a definition of software health and a structural equation model in the field of blockchain software health. This model can be used as a starting point for future research in this area and can help to guide the development of more comprehensive models of software health. A clear next step as stated in Section 11.4 is extension beyond blockchain OSS.

With regards to research into blockchain software, the nature of open source contributions allows developers to self-select projects that have an ideological fit (Smirnova et al., 2022). This has implications for the wider software industry, as it suggests that developers are more likely to contribute to projects that align with their personal beliefs and values. In the case of blockchain-based projects, for example, research shows that developers are more likely to cite motives for contributing based on a "Bitcoin ideology" than developers in non-blockchain domains (Bosu et al., 2019; Hars & Ou, 2001). This suggests that the blockchain industry may attract developers with a particular set of values and beliefs, which could guide newcomers looking to contribute to blockchain open source software. For example, project managers should be mindful of their project's ideological framing to attract suitable talent.

In addition to ideological factors, blockchain-based projects often rely on token incentives to motivate and reward developers. The suggestion here is that incentive-based participation may be more effective than purely voluntary contributions. While most OSS projects are built on voluntary contributions, grants, and scholarships, blockchain-based projects have the additional incentive of compensation through the token economy, either directly or indirectly. This creates a link between the quality of a developer's contribution and a potential financial reward, which may encourage more developers to contribute and improve the overall quality of the project.

The findings of this research can also extend to the realm of policy within open source communities. As the study operationalises software health metrics, this opens the door for standardisation within the OSS ecosystem. Governance bodies could potentially adopt these validated metrics as benchmarks for project health, making it easier for stakeholders to evaluate projects.[1] This could also affect the allocation of community or governmental grants and pave the way for more transparent, merit-based resource distribution. Furthermore, the insights regarding ideological alignment and token-based incentives could guide policy around contributor recruitment and retention, fostering a more cohesive and motivated contributor base.

In summary, this study serves multiple functions in its practical applicability. It offers a systematic and reproducible framework for assessing OSS health, thus contributing to the nascent academic discourse in this domain. The study particularly targets stakeholders ranging from current and prospective contributors to project managers and researchers, by operationalising latent constructs that facilitate resource allocation and strategic decision making.

11.4 Future Work

The list of second layer hypotheses presented in Section 10.5.2 discusses potential applications the framework can be used for to advance research. This list presents some of the directions future research can take. Other avenues of next steps involve broadening the scope of data applicability, expanding the time horizon, and returning to the issue of blockchain scaling.

11.4.1 Diversify the Data

The next step is to validate the work outside the blockchain domain. A good structural model (and measurement model) allows for generalisability, which can be tested across different OSS

1. This is broadly the goal of the CHAOSS project, "focused on creating metrics, metrics models, and software to better understand open source community health on a global scale." See https://chaoss.community/about-chaoss/

industries, such as mobile, web, tools, finance, and others. By testing the applicability of a structural model across multiple industry domains, researchers can assess the robustness and generalisability of the model, as well as identify any industry-specific factors that may impact software health. This can help to ensure that the model is widely applicable and can provide useful insights for practitioners across a range of collaborative software engineering contexts.

Beyond validating a sound structural model across industries, the direct application of software projects assessed against the model as a predictor of health is a long term goal. By using SEM to model the relationships between these endogenous concepts of *Interest*, *Engagement*, and *Robustness*, it is possible to make predictions about health based on the model. This can be particularly useful in the software development process, as it allows developers to identify which factors are most important for achieving desirable outcomes and to adjust their processes accordingly, perhaps even identifying successful projects.

Although the theoretical framework is based on the type of being able to explain and predict, the predicative capabilities need further time and data to be assessed. As mentioned, expanding the field of applicability, for the framework and SEM model, beyond blockchain could first target other open source industries.

11.4.2 Diversify the Time

Figure 4.1 outlines the research process where the time horizons for the present study are chosen as a cross-section. A time series analysis could be conducted to determine health and detect health issues in close to real time. This involves examining patterns and trends over time, in order to identify potential issues or areas of concern before they become major problems. By tracking key metrics related to software health, such as interest, engagement, and robustness, researchers can gain a better understanding of how these metrics change over time and how they are influenced by various factors. This can inform practitioners, enabling them to make more informed decisions and take proactive steps to address potential issues before they become critical.

Studies based on temporal results can also help software developers to determine the health of a project before contributing their personal time and resources. By evaluating the health of a project, developers can make informed decisions about whether to contribute to a project and how best to allocate their effort.

11.4.3 Returning to Blockchain Scaling

Blockchain scaling is left behind at the conclusion to RQ1 (see Section 10.1) with the link to consensus methods when the research proceeded to focus on health. In the intervening time

of this work, scaling is still an active area of blockchain research, with much of the effort focussing on layer 2 solutions such as the Lightning Network on Bitcoin, or rollup and zero-knowledge based methods on Ethereum. It may be that scaling presents as an ongoing issue with no writ solution available to be implemented by software teams, no matter how healthy the project. Rather, there is an inherent human nature to scale all systems they interact with, that is felt more prominently in popular contemporary systems, in this case blockchain networks. With regards to future work, blockchain scaling is its own area of distributed systems research, yet should be included in the discussion of software health as, is the hope of this researcher, lessons learned by investigating general interest, developer engagement, and software robustness are broadly applicable.

11.5 Conclusion

The thesis tackles the challenge of creating a theoretical framework to assess software health, specifically in the context of open source software blockchain projects. The motivation for this research is derived from the blockchain trilemma, a concept that highlights the balance between decentralisation, security, and scalability. By examining consensus methods and digging into the repositories where such algorithms are housed, the study conducts a comprehensive analysis of metrics that can determine health. The research not only addresses the question of what constitutes healthy software but also provides a framework for analysing software health.

The constructed framework serves as a contribution to the fields of both Information Systems and blockchain software. It offers practical ramifications not just for academic researchers, but also for practitioners, developers, project managers, and policymakers, essentially serving myriad stakeholders.

As for future avenues of exploration, the validation of the proposed model through its application in adjacent industries and an extension in its temporal scope stand out as clear extensions of this work. These could serve to broaden the empirical robustness of the model, further accentuating its utility.

References

Abdulhassan Alshomali, M. A. (2018). *Open source software GitHub ecosystem: A SEM approach* (Doctoral dissertation, James Cook University). https://doi.org/10.25903/5c3eb27776753

Abraham, I., Malkhi, D., Nayak, K., Ren, L., & Spiegelman, A. (2016). *Solida: A blockchain protocol based on reconfigurable Byzantine consensus.* arXiv. https://doi.org/10.48550/arXiv.1612.02916

Aguinis, H., Gottfredson, R. K., & Joo, H. (2013). Best-practice recommendations for defining, identifying, and handling outliers. *Organizational Research Methods*, *16*(2), 270–301. https://doi.org/10.1177/1094428112470848

Alavi, M., Visentin, D. C., Thapa, D. K., Hunt, G. E., Watson, R., & Cleary, M. (2020). Chi-square for model fit in confirmatory factor analysis. *Journal of Advanced Nursing*, *76*(9), 2209–2211. https://doi.org/10.1111/jan.14399

Alcaraz-Corona, S., Mata, J. L. C., & Torres-Castillo, F. (2019). Exploratory factor analysis for software development projects in Mexico. *Statistics, Optimization and Information Computing*, *7*(1), 85–96. https://doi.org/10.19139/soic.v7i1.512

Al-Houmaily, Y. J., & Samaras, G. (2016). Three-phase commit. In L. Liu & M. T. Özsu (Eds.), *Encyclopedia of database systems* (pp. 1–7). Springer. https://doi.org/10.1007/978-1-4899-7993-3_714-2

Alpern, B., & Schneider, F. B. (1985). Defining liveness. *Information Processing Letters*, *21*(4), 181–185. https://doi.org/10.1016/0020-0190(85)90056-0

Anderson, R. (2020). Distributed systems. In *Security engineering: A guide to building dependable distributed systems* (3rd ed., pp. 224–250). Wiley. https://doi.org/10.1002/9781119644682.ch7

Anthes, G. (2016). Open source software no longer optional. *Communications of the ACM*, *59*(8), 15–17. https://doi.org/10.1145/2949684

Antonopoulos, A. M. (2017). *Mastering Bitcoin : Programming the open blockchain.* O'Reilly.

Apache ZooKeeper. (2019, December 1). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Apache_ZooKeeper&oldid=992755046

Arantes, F. L., & Freire, F. M. P. (2011). Aspects of an open source software sustainable life cycle. *Open Source Systems: Grounding Research - 7th IFIP WG 2.13 International Conference*, 325–329. https://doi.org/10.1007/978-3-642-24418-6_26

Arksey, H., & O'Malley, L. (2005). Scoping studies: Towards a methodological framework. *International Journal of Social Research Methodology: Theory and Practice*, *8*(1), 19–32. https://doi.org/10.1080/1364557032000119616

Arya, A., Brown, C., & Pike, R. (2022). *Open source project criticality score* (Version 1.0.7) [Software]. https://github.com/ossf/criticality_score

Attiya, H., & Welch, J. (2004). *Distributed computing: Fundamentals, simulations and advanced topics* (2nd ed.). Wiley. https://doi.org/10.1002/0471478210

Atzei, N., Bartoletti, M., & Cimoli, T. (2017). A Survey of Attacks on Ethereum Smart Contracts (SoK). In M. Maffei & M. Ryan (Eds.), *Principles of security and trust* (pp. 164–186). Springer. https://doi.org/10.1007/978-3-662-54455-6_8

Bach, L., Mihaljevic, B., & Zagar, M. (2018). Comparative analysis of blockchain consensus algorithms. *MIPRO*, 1545–1550. https://doi.org/10.23919/MIPRO.2018.8400278

Back, A. (2002). Hashcash - A denial of service counter-measure. http://www.hashcash.org/

Bailey, K. D. (1994). Typologies and taxonomies: An introduction to classification techniques. *Quantitative Applications in the Social Sciences*, *101*. https://us.sagepub.com/en-us/nam/book/typologies-and-taxonomies

Bano, S., Al-Bassam, M., & Danezis, G. (2017). The road to scalable blockchain designs functional components of a blockchain. *;login:*, *42*(4), 31–36. https://www.usenix.org/publications/login/winter2017/bano

Bano, S., Sonnino, A., Al-Bassam, M., Azouvi, S., McCorry, P., Meiklejohn, S., & Danezis, G. (2017). *SoK: Consensus in the age of blockchains*. arXiv. https://doi.org/10.48550/arXiv.1711.03936

Baran, P. (1964). *On distributed communications: I. introduction to distributed communications networks*. RAND Corporation. https://www.rand.org/pubs/research_memoranda/RM3420.html

Barcomb, A., Stol, K.-J., Riehle, D., & Fitzgerald, B. (2019). Why do episodic volunteers stay in floss communities? *41st International Conference on Software Engineering*. http://hdl.handle.net/10468/7248

Bartlett, M. (1950). Tests of significance in factor analysis. *British Journal of Psychology*, *3*(2), 77–85.

Bartoletti, M., & Pompianu, L. (2017). An analysis of Bitcoin OP_RETURN metadata. https://arxiv.org/abs/1702.01024

Baudet, M., Ching, A., Chursin, A., Danezis, G., Garillot, F., Li, Z., … Sonnino, A. (2019). *State machine replication in the Libra blockchain*. https://developers.diem.com/docs/technical-papers/state-machine-replication-paper/

Bauer, M. W. (2000). Classical content analysis: A review. In *Qualitative researching with text, image and sound* (pp. 132–151). SAGE. https://doi.org/10.4135/9781849209731

Bawack, R. E., Wamba, S. F., & Carillo, K. D. A. (2021). A framework for understanding artificial intelligence research: Insights from practice. *Journal of Enterprise Information Management*, *34*(2), 645–678. https://doi.org/10.1108/JEIM-07-2020-0284

Bayonne, E., Marin-Garcia, J. A., & Alfalla-Luque, R. (2020). Partial least squares (PLS) in operations management research: Insights from a systematic literature review. *Journal of Industrial Engineering and Management*, *13*(3), 565–597. https://doi.org/10.3926/JIEM.3416

Beck, R., Weber, S., & Gregory, R. W. (2013). Theory-generating Design Science research. *Information Systems Frontiers*, *15*(4), 637–651. https://doi.org/10.1007/s10796-012-9342-4

Bellinger, C., Drummond, C., & Japkowicz, N. (2018). Manifold-based synthetic oversampling with manifold conformance estimation. *Machine Learning*, *107*(3), 605–637. https://doi.org/10.1007/s10994-017-5670-4

Benet, J., & Greco, N. (2018). *Filecoin: A decentralized storage network*. https://filecoin.io/filecoin.pdf

Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., & Virza, M. (2014). Zerocash: Decentralized anonymous payments from Bitcoin. *2014 IEEE Symposium on Security and Privacy*, 459–474. https://doi.org/10.1109/SP.2014.36

Bessani, A., Sousa, J. J., & Alchieri, E. E. P. (2014). State machine replication for the masses with BFT-SMART. *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 355–362. https://doi.org/10.1109/DSN.2014.43

Bharati, P., & Chaudhury, A. (2006). Product customization on the web: An empirical study of factors impacting choiceboard user satisfaction. *Information Resources Management Journal*, *19*(2), 69–81. https://doi.org/10.4018/irmj.2006040105

Bier, J. (2021). *The blocksize war: The battle over who controls Bitcoin's protocol rules*. (n.p.).

Blocki, J., & Zhou, H.-S. (2016). Designing proof of human-work puzzles for cryptocurrency and beyond. *Theory of Cryptography*, 517–546. https://doi.org/10.1007/978-3-662-53644-5_20

Bollen, K. A., & Stine, R. A. (1992). Bootstrapping goodness-of-fit measures in structural equation models. *Sociological Methods and Research*, *21*(2), 205. https://doi.org/10.1177/0049124192021002004

Borges, H., Hora, A., & Valente, M. T. (2017). Understanding the factors that impact the popularity of GitHub repositories. *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 334–344. https://doi.org/10.1109/ICSME.2016.31

Bosu, A., Iqbal, A., Shahriyar, R., & Chakroborty, P. (2019). Understanding the motivations, challenges and needs of blockchain software developers: A survey. *Empirical Software Engineering*, *24*, 2636–2673. https://doi.org/10.1007/s10664-019-09708-7

Boulding, K. E. (1980). *Illustrating economics: Beasts, ballads and aphorisms*. Routledge. https://doi.org/10.4324/9780203789162

Box, G. E. P. (1979). Robustness in the strategy of scientific model building. In R. L. Launer & G. N. Wilkinson (Eds.), *Robustness in statistics* (pp. 201–236). Academic Press.

Brewer, E. A. (2000). Towards robust distributed systems (abstract). *PODC '00: Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*. https://doi.org/10.1145/343477.343502

Brock, D. C. (2023, Mar). 50 years later, we're still living in the Xerox Alto's world. *IEEE Spectrum*. https://spectrum.ieee.org/xerox-alto

Brown, D. (2010). *Standards for efficient cryptography 2 (SEC 2)* (Vol. 2). http://www.secg.org/sec2-v2.pdf

Brown, Z. (2018, Jul 27). *A git origin story*. Linux Journal. https://www.linuxjournal.com/content/git-origin-story

Buterin, V. (2013). *A next-generation smart contract and decentralized application platform*. https://ethereum.org/en/whitepaper/

Buterin, V. (2016, November 18). *Sharding FAQ*. https://github.com/ethereum/wiki/wiki/Sharding-FAQ/e46274eebd62fc06ec8523279d0054fafb15ba85

Buterin, V., & Griffith, V. (2017). *Casper the friendly finality gadget*. arXiv. https://doi.org/10.48550/arXiv.1710.09437

Cachin, C. (2010). State machine replication with Byzantine faults. In *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)* (Vol. 5959 LNCS, pp. 169–184). https://doi.org/10.1007/978-3-642-11294-2_9

Cachin, C., & Vukolić, M. (2017). *Blockchain consensus protocols in the wild*. arXiv. https://arxiv.org/abs/1707.01873

Camenisch, J., Hohenberger, S., & Lysyanskaya, A. (2005). *Compact e-cash*. Cryptology ePrint Archive. https://eprint.iacr.org/2005/060

Campanelli, A. S., Camilo, R. D., & Parreiras, F. S. (2018). The impact of tailoring criteria on agile practices adoption: A survey with novice agile practitioners in Brazil. *Journal of Systems and Software*, *137*, 366–379. https://doi.org/10.1016/j.jss.2017.12.012

Capko, D., Vukmirović, S., & Nedić, N. (2022). State of the art of zero-knowledge proofs in blockchain. *2022 30th Telecommunications Forum (TELFOR)*. https://doi.org/10.1109/TELFOR56187.2022.9983760

Casino, F., Dasaklis, T. K., & Patsakis, C. (2019). A systematic literature review of blockchain-based applications: Current status, classification and open issues. *Telematics and Informatics*, *36*, 55–81. https://doi.org/10.1016/J.TELE.2018.11.006

Castro, M., & Liskov, B. (1999). Practical Byzantine fault tolerance. *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, *99*, 173–186. https://doi.org/10.5555/296806.296824

Cepeda-Carrion, G., Cegarra-Navarro, J.-G., & Cillo, V. (2018). Tips to use partial least squares structural equation modelling (PLS-SEM) in knowledge management. *Journal of Knowledge Management*, *23*(1), 67–89. https://doi.org/10.1108/JKM-05-2018-0322

Chakravorty, A., & Rong, C. (2017). Ushare: User controlled social media based on blockchain. *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication, IMCOM 2017*. https://doi.org/10.1145/3022227.3022325

Chalaemwongwan, N., & Kurutach, W. (2018). State of the art and challenges facing consensus protocols on blockchain. *2018 International Conference on Information Networking (ICOIN)*, 957–962. https://doi.org/10.1109/ICOIN.2018.8343266

Chandrachooodan, G., & Radhika, R. (2022). Exploratory factor analysis for identifying the factors affecting adoption of project management methodology in Information Systems development in Kerala. *AIP Conference Proceedings*, *2520*. https://doi.org/10.1063/5.0102981

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., … Gruber, R. E. (2006). *Bigtable: A distributed storage system for structured data*. Google. https://ai.google/research/pubs/pub27898

Chaudhry, N., & Yousaf, M. M. (2018). Consensus algorithms in blockchain: Comparative analysis, challenges and opportunities. *2018 12th International Conference on Open Source Systems and Technologies (ICOSST)*, 54–63. https://doi.org/10.1109/ICOSST.2018.8632190

Chaum, D. (1983). Blind Signatures for Untraceable Payments. *Advances in Cryptology*, 199–203.

Chaum, D., Fiat, A., & Naor, M. (1988). Untraceable Electronic Cash. *Advances in Cryptology — CRYPTO' 88*, 319–327. https://doi.org/10.1007/0-387-34799-2_25

Chen, C. C., Liu, J. Y. C., & Chen, H. G. (2011). Discriminative effect of user influence and user responsibility on information system development processes and project management. *Information and Software Technology*, *53*(2), 149–158. https://doi.org/10.1016/j.infsof.2010.10.001

Chen, F., Curran, P. J., Bollen, K. A., Kirby, J., & Paxton, P. (2008). Inequality convergence: How sensitive are results to the choice of data? *Sociological Methods & Research*, *36*(4), 462–494. https://doi.org/10.1177/0049124108314720

Chengalur-Smith, I., Sidorova, A., & Daniel, S. (2010). Sustainability of free/libre open source projects: A longitudinal study. *Journal of the Association for Information Systems*, *11*(11), 657–683. https://doi.org/10.17705/1jais.00244

Chow, T., & Cao, D. B. (2008). A survey study of critical success factors in agile software projects. *Journal of Systems and Software*, *81*(6), 961–971. https://doi.org/10.1016/j.jss.2007.08.020

Christidis, K., & Devetsikiotis, M. (2016). Blockchains and smart contracts for the internet of things. *IEEE Access*, *4*, 2292–2303. https://doi.org/10.1109/ACCESS.2016.2566339

Christin, N. (2013). Traveling the Silk Road: A measurement analysis of a large anonymous online marketplace. *Proceedings of the 22nd International Conference on World Wide Web*, 213–223. https://doi.org/10.1145/2488388.2488408

Clark, M. (2018). *Structural equation modeling*. https://m-clark.github.io/sem/

Coelho, J., Valente, M. T., Silva, L. L., & Hora, A. (2018). Why we engage in FLOSS: Answers from core developers. *Proceedings - International Conference on Software Engineering*, 114–121. https://doi.org/10.1145/3195836.3195848

Conti, M., Sandeep, K. E., Lal, C., & Ruj, S. (2018). A survey on security and privacy issues of bitcoin. *IEEE Communications Surveys and Tutorials*, *20*(4), 3416–3452. https://doi.org/10.1109/COMST.2018.2842460

Cooper, G. E., & Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, *9*, 309–347. https://doi.org/10.1007/BF00994110

Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J., … Woodford, D. (2012). Spanner: Google's globally-distributed database. *OSDI 2012*, 1–14. http://research.google.com/archive/spanner-osdi2012.pdf

Costanza, R. (1992). Toward an operational definition of ecosystem health. In R. Costanza, B. Norton, & B. J. Haskell (Eds.), (pp. 239–256). Island.

Crisman, D., Logan, J., & Bansal, M. (2021). Open-source software: Risks and rewards. *The Licensing Journal*, *41*(10), 1–6. https://www.morganlewis.com/pubs/2021/06/insight-best-practices-for-leveraging-open-source-software-and-mitigating-its-risks

Croman, K., Decker, C., Eyal, I., Efe Gencer, A., Juels, A., Kosba, A., … Wattenhofer, R. (2016). On scaling decentralized blockchains (a position paper). *International Conference on Financial Cryptography and Data Security*, 106–125. http://fc16.ifca.ai/bitcoin/papers/CDE+16.pdf

Cronbach, L. J. (1951). Coefficient alpha and the internal structure of tests. *Psychometrika*, *16*, 297–334. https://doi.org/10.1007/BF02310555

Crowston, K., & Howison, J. (2011). Floss project effectiveness measures. In H. Benbya & N. Belbaly (Eds.), *Successful oss project design and implementation: Requirements, tools, social designs and reward structures* (pp. 149–168). https://doi.org/10.4324/9781315611235

Crowston, K., Howison, J., & Annabi, H. (2006). Information systems success in free and open source software development: Theory and measures. *Software Process Improvement and Practice*, *11*(2), 123–148. https://doi.org/10.1002/spip.259

Cushing, B. E. (1990). Frameworks, paradigms, and scientific research in Management Information Systems. *Journal of Information Systems*, *4*(2), 38–59.

Danezis, G., & Meiklejohn, S. (2016). Centrally banked cryptocurrencies. *NDSS '16*. https://doi.org/10.14722/ndss.2016.23187

Daniel, S., Agarwal, & Stewart. (2013). The effects of diversity in global, distributed collectives: A study of open source project success. *Information Systems Research*, *24*(2), 312–333. https://doi.org/10.1287/isre.1120.0435

Davis, K., Drey, N., & Gould, D. (2009). What are scoping studies? A review of the nursing literature. *International journal of nursing studies*, *46*(10), 1386–400. https://doi.org/10.1016/j.ijnurstu.2009.02.010

De, N. (2022, February 1). *Diem confirms shutdown as Silvergate acquires the project's assets.* CoinDesk. https://www.coindesk.com/business/2022/01/31/silvergate-bank-confirms-diem-tech-acquisition/

Decandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., … Vogels, W. (2007). *Dynamo: Amazon's highly available key-value store.* https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf

Decker, C. (2020, May 05). *Comprehensive academic Bitcoin research archive.* https://cdecker.github.io/btcresearch/

Decker, C., Seidel, J., & Wattenhofer, R. (2016). Bitcoin Meets Strong Consistency. *Proceedings of the 17th International Conference on Distributed Computing and Networking*, 13:1—-13:10. https://doi.org/10.1145/2833312.2833321

Decker, C., & Wattenhofer, R. (2015). A fast and scalable payment network with bitcoin duplex micropayment channels. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9212*, 3–18. https://doi.org/10.1007/978-3-319-21741-3_1

DeLone, W. H., & McLean, E. R. (1992). Information Systems success: The quest for the dependent variable. *Information Systems Research*, *3*(1), 60–95. https://doi.org/10.1287/isre.3.1.60

Dhungana, D., Groher, I., Schludermann, E., & Biffl, S. (2010). Software ecosystems vs. natural ecosystems: Learning from the ingenious mind of nature. *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, 96–102. https://doi.org/10.1145/1842752.1842777

Dinh, T. T. A., Liu, R., Zhang, M., Chen, G., Ooi, B. C., & Wang, J. (2018). Untangling blockchain: A data processing view of blockchain systems. *IEEE Transactions on Knowledge and Data Engineering*, *30*(7), 1366–1385. https://doi.org/10.1109/TKDE.2017.2781227

Dinh, T. T. A., Wang, J., Chen, G., Liu, R., Ooi, B. C., & Tan, K.-L. (2017). BLOCKBENCH: A framework for analyzing private blockchains. *Proceedings of the 2017 ACM International Conference on Management of Data*, 1085–1100. https://doi.org/10.1145/3035918.3064033

Dow, K. E., Wong, J., Jackson, C., & Leitch, R. A. (2008). A comparison of structural equation modeling approaches: The case of user acceptance of Information Systems. *Journal of Computer Information Systems*, *48*(4), 106–114. https://doi.org/10.1080/08874417.2008.11646040

Dubé, L., & Paré, G. (2003). Rigor in Information Systems positivist case research: Current practices, trends, and recommendations. *MIS Quarterly*, *27*(4), 597–636. https://doi.org/10.2307/30036550

Dueñas, S., Cosentino, V., Robles, G., & González-Barahona, J. M. (2018). Perceval: software project data at your will. *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, 1–4. https://doi.org/10.1145/3183440.3183475

Dunphy, P., & Petitcolas, F. A. (2018). A first look at identity management schemes on the blockchain. *IEEE Security and Privacy*, *16*(4), 20–29. https://doi.org/10.1109/MSP.2018.3111247

Dwork, C., & Naor, M. (1992). Pricing Via Processing or Combatting Junk Mail. *CRYPTO '92 Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, 11. https://dl.acm.org/citation.cfm?id=705669

Edge, J. (2013, March 6). *ELC: SpaceX lessons learned.* https://lwn.net/Articles/540368/

Einstein, A. (2011). *Out of my later years: The scientist, philosopher, and man portrayed through his own words.* Philosophical Library/Open Road.

Elo, S., & Kyngäs, H. (2008). The qualitative content analysis process. *Journal of Advanced Nursing*, *62*(1), 107–115. https://doi.org/10.1111/j.1365-2648.2007.04569.x

Eyal, I., & Sirer, E. G. (2014). Majority is not enough: Bitcoin mining is vulnerable. *Financial Cryptography and Data Security*, 436–454. https://doi.org/10.1007/978-3-662-45472-5_28

Fabrigar, L., & Wegener, D. (2012). *Exploratory factor analysis.* Oxford University Press. https://doi.org/10.1093/acprof:osobl/9780199734177.001.0001

Fang, Y., & Neufeld, D. (2008). Understanding sustained participation in open source software projects. *Journal of Management Information Systems*, *25*(4), 9–50. https://doi.org/10.2753/MIS0742-1222250401

Favell, A. (2020, February 4). *The history of git: The road to domination in software version control.* https://www.welcometothejungle.com/en/articles/btc-history-git

Federal Aviation Administration. (2012). *Flight-critical systems design assurance* (Final Report No. DOT/FAA/AR-11/28). https://www.tc.faa.gov/its/worldpac/techrpt/ar11-28.pdf

Fidell, L. S., & Tabachnick, B. G. (2003). Preparatory data analysis. In *Handbook of psychology* (pp. 115–141). Wiley. https://doi.org/10.1002/0471264385.wei0205

Finch, W. H. (2020a). *Exploratory factor analysis.* SAGE. https://doi.org/10.4135/9781544339900

Finch, W. H. (2020b). Using fit statistic differences to determine the optimal number of factors to retain in an exploratory factor analysis. *Educational and Psychological Measurement*, *80*(2), 217–241. https://doi.org/10.1177/0013164419865769

Fischer, M. J., Lynch, N. A., & Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Association for Computing Machinery*, *32*(2), 374–382. https://doi.org/10.1145/3149.214121

Fitzgerald, B. (2006). The transformation of open source software. *MIS Quarterly*, *30*(3), 587–598. https://doi.org/10.2307/25148740

Flick, U. (2014). *The SAGE handbook of qualitative data analysis* (4th ed.). SAGE. https://doi.org/10.4135/9781071802779

Franco-Bedoya, O., Ameller, D., Costal, D., & Franch, X. (2014). Queso: a quality model for open source software ecosystems. *9th International Conference on Software Engineering and Applications (ICSOFT-EA)*, 209–221. https://doi.org/10.5220/0004993702090221

Garay, J., Kiayias, A., & Leonardos, N. (2015). The Bitcoin backbone protocol: Analysis and applications. In E. Oswald & M. Fischlin (Eds.), *Advances in cryptology - eurocrypt 2015* (pp. 281–310). Springer.

Garay, J. A., & Kiayias, A. (2019). *SoK: A consensus taxonomy in the blockchain era.* Cryptology ePrint Archive. https://eprint.iacr.org/2018/754.pdf

Garay, J. A., Kiayias, A., & Leonardos, N. (2019). *The Bitcoin backbone protocol: Analysis and applications.* Cryptology ePrint Archive. https://eprint.iacr.org/2014/765.pdf

Garay, J. A., Kiayias, A., & Leonardos, N. (2020). *Full analysis of Nakamoto consensus in bounded-delay networks.* Cryptology ePrint Archive. https://eprint.iacr.org/2020/277.pdf

Garfinkel, B. (2018). *Recent developments in cryptography and possible long-run consequences.* https://drive.google.com/file/d/0B0j9LKC65n09aDh4RmEzdlloT00/view

Garoffolo, A., Kaidalov, D., & Oliynykov, R. (2020). Zendoo: a zk-SNARK verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains. *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, 1257–1262. https://doi.org/10.1109/ICDCS47774.2020.00161

Gemini Trust Company. (2019). *The Gemini dollar : A regulated stable value coin.* https://gemini.com/static/dollar/gemini-dollar-whitepaper.pdf

Gericke, A., Klesse, M., Winter, R., & Wortmann, F. (2010). Success factors of application integration: An exploratory analysis. *Communications of the Association for Information Systems*, *27*. https://doi.org/10.17705/1cais.02737

Ghapanchi, A. (2015). Investigating the interrelationships among success measures of open source software projects. *Journal of Organizational Computing and Electronic Commerce*, *25*(1), 28–46. https://doi.org/10.1080/10919392.2015.990775

Ghapanchi, A., Aurum, A., & Low, G. (2011). A taxonomy for measuring the success of open source software projects. *First Monday*, *16*(8). https://doi.org/10.5210/fm.v16i8.3558

Gilad, Y., Hemo, R., Micali, S., Vlachos, G., & Zeldovich, N. (2017). Algorand: Scaling Byzantine agreements for cryptocurrencies. *Proceedings of the 26th Symposium on Operating Systems Principles. ACM, 2017*, 51–68. https://doi.org/10.1145/3132747.3132757

Gilbert, S., & Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, *33*(2), 51. https://doi.org/10.1145/564585.564601

GitHub. (2021). *The 2021 state of the octoverse.* https://octoverse.github.com/

GitHub. (2023). *Octoverse: The state of open source and rise of AI in 2023.* https://github.blog/2023-11-08-the-state-of-open-source-and-ai/

Glaser, F., & Bezzenberger, L. (2015). Beyond cryptocurrencies - A taxonomy of decentralized consensus systems. *ECIS 2015 Completed Research Papers*, 1–18. https://doi.org/10.18151/7217326

Goeminne, M., & Mens, T. (2013). Analyzing ecosystems for open source software developer communities. In S. Jansen (Ed.), *Software ecosystems: Analyzing and managing business networks in the software industry* (pp. 247–275). https://doi.org/10.4337/9781781955635.00021

Goggins, S., Lumbard, K., & Germonprez, M. (2021). Open source community health: Analytical metrics and their corresponding narratives. *Proceedings - 2021 IEEE/ACM 4th International Workshop on Software Health in Projects, Ecosystems and Communities, SoHeal 2021*, 25–33. https://doi.org/10.1109/SOHEAL52568.2021.00010

Goldkuhl, G. (2012). Pragmatism vs interpretivism in qualitative Information Systems research. *European Journal of Information Systems*, *21*(2), 135–146. https://doi.org/10.1057/ejis.2011.54

Gonzalez-Barahona, J. M. (2021). A brief history of free, open source software and its communities. *Computer*, *54*(2), 75–79. https://doi.org/10.1109/MC.2020.3041887

Gousios, G., Pinzger, M., & Deursen, A. V. (2014). An exploratory study of the pull-based software development model. *Proceedings - International Conference on Software Engineering*(1), 345–355. https://doi.org/10.1145/2568225.2568260

Gousios, G., Storey, M. A., & Bacchelli, A. (2016). Work practices and challenges in pull-based development: The contributor's perspective. *Proceedings - International Conference on Software Engineering*, 285–296. https://doi.org/10.1145/2884781.2884826

Grant, C., & Osanloo, A. (2014). Understanding, selecting, and integrating a theoretical framework in dissertation research: Creating the blueprint for your "house". *Administrative Issues Journal Education Practice and Research*, *4*(2). https://doi.org/10.5929/2014.4.2.9

Gray, J., & Lamport, L. (2006). Consensus on transaction commit. *ACM Transactions on Database Systems*, *31*(1), 133–160. https://doi.org/10.1145/1132863.1132867

Gray, J. N. (1978). Notes on data base operating systems. In R. Bayer, R. M. Graham, & G. Seegmüller (Eds.), *Operating systems: An advanced course* (pp. 393–481). Springer. https://doi.org/10.1007/3-540-08755-9_9

Gregor, S. (2006). The nature of theory in Information Systems. *MIS Quarterly*, *30*(3), 611–642. https://doi.org/10.2307/25148742

Grover, V., Jeong, S. R., & Segars, A. H. (1996). Information Systems effectiveness: The construct space of patterns and application. *Information & Management*, *31*, 177–191. https://doi.org/10.1016/S0378-7206(96)01079-8

Gu, Z., Wang, Y., Hua, Q.-S., & Lau, F. C. (2017). *Rendezvous in distributed systems: theory, algorithms and applications*. Springer. https://doi.org/10.1007/978-981-10-3680-4

Haber, S., & Stornetta, W. S. (1991). How to time-stamp a digital document. In A. J. Menezes & S. A. Vanstone (Eds.), *Advances in cryptology-crypto '90* (pp. 437–455). Springer. https://doi.org/10.1007/3-540-38424-3_32

Hadzilacos, V., & Toueg, S. (1993). Fault-tolerant broadcasts and related problems. In *Distributed systems* (2nd ed., p. 97–145). ACM Press; Addison-Wesley. https://doi.org/10.5555/302430.302435

Hair, J., Hollingsworth, C. L., Randolph, A. B., & Chong, A. Y. L. (2017). An updated and expanded assessment of PLS-SEM in Information Systems research. *Industrial Management and Data Systems*, *117*(3), 442–458. https://doi.org/10.1108/IMDS-04-2016-0130

Hair Jr., J. F., Black, W. C., Babin, B. J., & Anderson, R. E. (2014). *Multivariate data analysis* (7th ed.). Pearson.

Hair Jr., J. F., Hult, G. T. M., Ringle, C. M., Sarstedt, M., Danks, N. P., & Ray, S. (2021). *Partial least squares structural equation modeling (PLS-SEM) using R: A workbook*. Springer.

Hammett, R. (2002). Flight-critical distributed systems - design considerations. *Proceedings of the 21st Digital Avionics Systems Conference*, *2*, 13B3–13B3. https://doi.org/10.1109/DASC.2002.1053006

Hanke, T., Movahedi, M., & Williams, D. (2018). *DFINITY technology overview series, consensus system.* arXiv. https://arxiv.org/pdf/1805.04548

Harman, H. H. (1976). *Modern factor analysis* (3rd ed.). University of Chicago Press.

Hars, A., & Ou, S. (2001). Working for free? - Motivations of participating in open source projects. *Proceedings of the Hawaii International Conference on System Sciences*, 163. https://doi.org/10.1109/hicss.2001.927045

Hartigh, E. D., Visscher, W., & Tol, M. (2013). Measuring the health of a business ecosystem. *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry*, 221–246.

Hata, H., Novielli, N., Baltes, S., Kula, R. G., & Treude, C. (2022). GitHub Discussions: An exploratory study of early adoption. *Empirical Software Engineering*, *27*(1), 1–32. https://doi.org/10.1007/s10664-021-10058-6

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems research. *MIS Quarterly*, *28*(1), 75–105. https://doi.org/10.2307/25148625

Hofacker, C. F. (2007). *Mathematical marketing*. New South Network Services.

Howard, M. C. (2016). A review of exploratory factor analysis decisions and overview of current practices: What we are doing and how can we improve? *International Journal of Human-Computer Interaction*, *32*(1), 51–62. https://doi.org/10.1080/10447318.2015.1087664

Hu, Y., Zhang, J., Bai, X., Yu, S., & Yang, Z. (2016). Influence analysis of github repositories. *SpringerPlus*, *5*(1). https://doi.org/10.1186/s40064-016-2897-7

Iansiti, M., & Levien, R. (2004). Strategy as ecology. *Harvard Business Review*, *BR0403*, 68–78. https://hbr.org/2004/03/strategy-as-ecology

Iivari, J., Hirschheim, R., & Klein, H. K. (2000). A dynamic framework for classifying Information Systems development methodologies and approaches. *Journal of Management Information Systems*, *17*(3), 179–218. https://doi.org/10.1080/07421222.2000.11045656

Jabareen, Y. (2009). Building a conceptual framework: Philosophy, definitions, and procedure. *International Journal of Qualitative Methods*, *8*(4), 49–62. https://doi.org/10.1177/160940690900800406

Jansen, S. (2014). Measuring the health of open source software ecosystems: Beyond the scope of project health. *Information and Software Technology*, *56*(11), 1508–1519. https://doi.org/10.1016/j.infsof.2014.04.006

Jansen, S., Finkelstein, A., & Brinkkemper, S. (2009). A sense of community: A research agenda for software ecosystems. *31st International Conference on Software Engineering, New and Emerging Research Track*, 187–190. https://slingerjansen.files.wordpress.com/2009/04/ssnniericse.pdf

Johnson, P., Ekstedt, M., & Jacobson, I. (2012). Where's the theory for software engineering? *IEEE Software*, *25*(9). https://doi.org/10.1109/MS.2012.127

Kaiser, H. F. (1974). An index of factorial simplicity. *Psychometrika*, *39*(1), 31–36. https://doi.org/10.1007/BF02291575

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., & Damian, D. (2016). An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering*, *21*(5), 2035–2071. https://doi.org/10.1007/s10664-015-9393-5

Kaufmann, L., & Gaeckler, J. (2015). A structured review of partial least squares in supply chain management research. *Journal of Purchasing and Supply Management*, *21*(4), 259–272.

Kerlinger, F. N. (1973). *Foundations of behavioral research* (2nd ed.). Holt, Rinehart and Winston.

Khine, M. S. (Ed.). (2013). *Application of structural equation modeling in educational research and practice* (Vol. 7). Sense. https://doi.org/10.1007/978-94-6209-332-4

Kitchenham, B., & Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering*. https://www.researchgate.net/publication/302924724_Guidelines_for_performing_Systematic_Literature_Reviews_in_Software_Engineering

Klein, H. K., & Myers, M. D. (1999). A set of principles for conducting and evaluating interpretive field studies in Information Systems. *MIS Quarterly*, *23*(1), 67–93. https://doi.org/10.2307/249410

Kokoris-Kogias, E., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., & Ford, B. (2016). Enhancing bitcoin security and performance with strong consistency via collective signing. *25th USENIX Security Symposium*. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kogias

Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., & Ford, B. (2018). OmniLedger: A secure, scale-out, decentralized ledger via sharding. *Proceedings - IEEE Symposium on Security and Privacy*, 583–598. https://doi.org/10.1109/SP.2018.000-5

Kotzé, P., Van Der Merwe, A., & Gerber, A. (2015). Design science research as research approach in doctoral studies. *2015 Americas Conference on Information Systems, AMCIS 2015*, 1–14. https://aisel.aisnet.org/amcis2015/DSR/GeneralPresentations/3/

Kripalani, R. (2017, August 21). *An analysis of the licenses used by 75+ open source projects across 35 companies.* https://medium.com/@raulk/list-of-companies-and-popular-projects-by-the-open-source-licenses-they-use-35a53eaf1c80

Krippendorff, K. (2004). *Content analysis: An introduction to its methodology.* SAGE.

Lamport, L. (1977). Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, *SE-3*(2), 125–143.

Lamport, L. (1998). The part-time parliament. *ACM Transactions on Computer Systems*, *16*(2), 133–169. https://doi.org/10.1145/279227.279229

Lamport, L. (2006). Fast Paxos. *Distributed Computing*, *19*(2), 79–103. https://doi.org/10.1007/s00446-006-0005-x

Lamport, L., Shostak, R., & Pease, M. (1982). The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, *4*(3), 382–401. https://doi.org/10.1145/357172.357176

Lampson, B., & Sturgis, H. (1976). *Crash recovery in a distributed system.* Xerox Research Centre.

Langer, W. (2019, May 24). How to use Stata's SEM command with nonnormal data? A new nonnormality correction for the RMSEA, CFI and TLI. *Meeting of the German Stata Users Group at the Ludwig-Maximilians Universität*. https://www.stata.com/meeting/germany19/slides/germany19_Langer.pdf

Laurent, A. M. S. (2004). *Understanding open source and free software licensing.* O'Reilly.

Lee, A., Carver, J. C., & Bosu, A. (2017). Understanding the impressions, motivations, and barriers of one-time code contributors to FLOSS projects: A survey. *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering, ICSE 2017*, 187–197. https://doi.org/10.1109/ICSE.2017.25

Li, J., Loucks, W., Zhai, Y. I., & Zhong, T. (2018). *The national institute of standards and technology post-quantum cryptography.* https://csrc.nist.rip/publications/detail/nistir/8105/final

Lima, A., Rossi, L., & Musolesi, M. (2014). Coding together at scale: GitHub as a collaborative social network. *Proceedings of the 8th International Conference on Weblogs and Social Media, ICWSM 2014*, 295–304.

Lindman, J. (2021). What open source software research can teach us about public blockchain(s)?—lessons for practitioners and future research. *Frontiers in Human Dynamics*, 13. https://doi.org/10.3389/FHUMD.2021.642556

Lindsay, B. G., Selinger, P. G., Galtieri, C., Gray, J. N., Lorie, R., Price, T. G., … Wade, B. W. (1979). Notes on distributed databases. *IBM Tech. Rept. RJ2571.* https://dominoweb.draco.res.ibm.com/reports/RJ2571.pdf

Link, G. J., & Germonprez, M. (2018). Assessing open source project health. *AMCIS 2018 Proceedings.* https://aisel.aisnet.org/amcis2018/Openness/Presentations/5

Loeliger, J., & McCullough, M. (2012). *Version control with git: Powerful tools and techniques for collaborative software development.* O'Reilly.

Luu, L., Baweja, K., Narayanan, V., Gilbert, S., Zheng, C., & Saxena, P. (2016). A secure sharding protocol for open blockchains. *CCS '16*. https://doi.org/10.1145/2976749.2978389

MacCallum, R. C., & Austin, J. T. (2000). Applications of structural equation modeling in psychological research. *Annual Review of Psychology*, *51*(1), 201–226. https://doi.org/10.1146/annurev.psych.51.1.201

Manikas, K., & Hansen, K. M. (2013). Reviewing the health of software ecosystems - a conceptual framework proposal. *5th Workshop on Software Ecosystems (IWSECO).* https://api.semanticscholar.org/CorpusID:2454407

Marcot, B. G., & Penman, T. D. (2019). *Advances in Bayesian network modelling: Integration of modelling technologies* (Vol. 111). Elsevier. https://doi.org/10.1016/j.envsoft.2018.09.016

McDonald, R. P. (1999). *Test theory: A unified treatment.* Lawrence Erlbaum Associates.

McKusick, M. K. (1999). *Open sources: Voices from the open source revolution.* O'Reilly.

Merkle, R. C. (1980). Protocols for public key cryptosystems. *IEEE Symposium on Security and Privacy*, 122–134. http://www.merkle.com/papers/Protocols.pdf

Merton, R. K. (1957). *Social theory and social structure.* Free Press.

Metz, C. (2015, Mar 12). *How GitHub conquered Google, Microsoft, and everyone else.* Wired. https://www.wired.com/2015/03/github-conquered-google-microsoft-everyone-else/

Miller, A., Xia, Y., Croman, K., Shi, E., & Song, D. (2016). The honey badger of BFT protocols. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*, 31–42. https://doi.org/10.1145/2976749.2978399

Miller, C. (2022). *Chip war: The fight for the world's most critical technology.* Scribner.

Milovidov, A. (2020). *Everything you ever wanted to know about GitHub (but were afraid to ask).* https://ghe.clickhouse.tech/

Moore, J. F. (1993). Predators and prey: A new ecology of competition. *Harvard Business Review*, *71*(3), 75–86.

Morgan, D., & Guevara, H. (2008). (L. M. Given, Ed.). SAGE.

Mundfrom, D. J., Shaw, D. G., & Ke, T. L. (2005). Minimum sample size recommendations for conducting factor analyses. *International Journal of Testing*, *5*(2), 159–168. https://doi.org/10.1207/s15327574ijt0502_4

Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System.* https://bitcoin.org/bitcoin.pdf

Naparat, D., Cahalane, M., & Finnegan, P. (2015). *Healthy community and healthy commons: 'opensourcing' as a sustainable model of software production* (Vol. 19). https://doi.org/10.3127/ajis.v19i0.1221

Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2016). *Bitcoin and cryptocurrency technologies : A comprehensive introduction.* Princeton University Press. https://press.princeton.edu/titles/10908.html

Narayanan, A., & Clark, J. (2017). Bitcoin's academic pedigree. *Communications of the ACM*, *60*(12), 36–45. https://doi.org/10.1145/3132259

Nasir, M. H. N., & Sahibuddin, S. (2011). Critical success factors for software projects: A comparative study. *Scientific Research and Essays*, *6*(10), 2174–2186. https://doi.org/10.5897/sre10.1171

National Institute of Standards and Technology. (2012). *Secure hash standard (SHS).* Federal Information Processing Standards. https://doi.org/10.6028/NIST.FIPS.180-4

Negoita, B., Vial, G., Shaikh, M., & Labbe, A. (2019). Code forking and software development project sustainability: Evidence from github. *40th International Conference on Information Systems, ICIS 2019*. https://aisel.aisnet.org/icis2019/is_development/is_development/7

Nemec, M., Klinec, D., Svenda, P., Sekan, P., & Matyas, V. (2017). Measuring popularity of cryptographic libraries in internet-wide scans. *Proceedings of the 33rd Annual Computer Security Applications Conference*, 162–175. https://doi.org/10.1145/3134600.3134612

Neuendorf, K. (2017). *The content analysis guidebook.* SAGE. https://doi.org/10.4135/9781071802878

Nickerson, R. C., Varshney, U., & Muntermann, J. (2013). A method for taxonomy development and its application in Information Systems. *European Journal of Information Systems*, *22*(3), 336–359. https://doi.org/10.1057/ejis.2012.26

Nijsse, J., & Litchfield, A. (2020). A taxonomy of blockchain consensus methods. *Cryptography*, *4*(4), 1–15. https://doi.org/10.3390/cryptography4040032

Nijsse, J., & Litchfield, A. (2023a). Identifying developer engagement in open source software blockchain projects through factor analysis. *56th Hawaii International Conference on System Sciences*, 5333–5342. https://hdl.handle.net/10125/103285

Nijsse, J., & Litchfield, A. (2023b). *Towards a structural equation model of open source blockchain software health.* arXiv. https://doi.org/10.48550/arXiv.2310.20277

Nord, J. H., Koohang, A., & Paliszkiewicz, J. (2019). The internet of things: Review and theoretical framework. *Expert Systems with Applications*, *133*, 97–108. https://doi.org/10.1016/j.eswa.2019.05.014

Norman, G. R., & Streiner, D. L. (2008). *Biostatistics: the bare essentials* (3rd ed.). BC Decker.

Offermann, P., Levina, O., Schönherr, M., & Bub, U. (2009). Outline of a design science research process. *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology, DESRIST '09*. https://doi.org/10.1145/1555619.1555629

Okoli, C. (2015). A guide to conducting a standalone systematic literature review. *Communications of the Association for Information Systems*, *37*. https://doi.org/10.17705/1CAIS.03743

Okutan, A., & Yıldız, O. T. (2014). Software defect prediction using Bayesian networks. *Empirical Software Engineering*, *19*(1), 154–181. https://doi.org/10.1007/s10664-012-9218-8

Open Source Initiative. (2007). *GNU general public license version 3.* https://opensource.org/licenses/GPL-3.0

Open Source Initiative. (2023). *The open source definition.* https://opensource.org/osd/

Oppenheimer, A. (2004, January). *A history of macintosh networking.* http://www.opendoor.biz/nethistory/MacWorld2004/index.html

Orlikowski, W. J., & Iacono, C. S. (2001). Research commentary: Desperately seeking the "IT" in IT research - a call to theorizing the IT artifact. *Information Systems Research*, *12*(2), 121–134. https://doi.org/10.1287/isre.12.2.121.9700

Osman, K., & Baysal, O. (2021). Health is wealth: Evaluating the health of the bitcoin ecosystem in github. *IEEE/ACM 4th International Workshop on Software Health in Projects, Ecosystems and Communities (SoHeal).* https://doi.org/10.1109/SoHeal52568.2021.00007

Park, S., Alwen, J., Fuchsbauer, G., Gazi, P., & Pietrzak, K. (2015). *SpaceMint: A cryptocurrency based on proofs of space.* Cryptology ePrint Archive. https://eprint.iacr.org/2015/528.pdf

Pass, R., Seeman, L., & Shelat, A. (2017). Analysis of the blockchain protocol in asynchronous networks. In *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)* (Vol. 10211 LNCS, pp. 643–673). https://doi.org/10.1007/978-3-319-56614-6_22

Pass, R., & Shi, E. (2017). Hybrid consensus: Efficient consensus in the permissionless model. *31st International Symposium on Distributed Computing (DISC 2017)*, *91*(39). https://doi.org/10.1212/WNL.0000000000001930

Pass, R., & Shi, E. (2018). Thunderella: Blockchains with optimistic instant confirmation. In *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)* (Vol. 10821 LNCS, pp. 3–33). Springer. https://doi.org/10.1007/978-3-319-78375-8_1

PayPal. (2018). *Paypal reports fourth quarter and full year 2017 results.* https://newsroom.paypal-corp.com/2018-01-31-PayPal-Reports-Fourth-Quarter-and-Full-Year-2017-Results

Pease, M., Shostak, R., & Lamport, L. (1980). Reaching Agreement in the Presence of Faults. *Journal of the ACM (JACM)*, *27*(2), 228–234. https://doi.org/10.1145/322186.322188

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems research. *Source: Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Pinto, A. M. (2020). An introduction to the use of zk-SNARKs in blockchains. *Mathematical Research for Blockchain Economy*, 233–249.

Pinto, G., Steinmacher, I., & Gerosa, M. A. (2016). More common than you think: An in-depth study of casual contributors. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016*, *1*(1), 112–123. https://doi.org/10.1109/SANER.2016.68

Poba-Nzaou, P., & Uwizeyemungu, S. (2019). Worries of open source projects' contributors: Patterns, structures and engagement implications. *Computers in Human Behavior*, *96*, 174–185. https://doi.org/10.1016/j.chb.2019.02.005

Poon, J., & Buterin, V. (2017). *Plasma : Scalable autonomous smart contracts scalable multi-party computation.* https://plasma.io/plasma.pdf

Poon, J., & Dryja, T. (2016). *The Bitcoin lightning network: Scalable off-chain instant payments.* https://lightning.network/lightning-network-paper.pdf

Pop, C., Cioara, T., Antal, M., Anghel, I., Salomie, I., & Bertoncini, M. (2018). Blockchain based decentralized management of demand response programs in smart energy grids. *Sensors*, *18*(1). https://doi.org/10.3390/s18010162

Popper, K. (2002). *The logic of scientific discovery.* Routledge Classics.

Qian, Z., & Bock, G.-W. (2005). An empirical study on measuring the success of knowledge repository systems. *Proceedings of the 38th Annual Hawaii International Conference on System Sciences.* https://doi.org/10.1109/HICSS.2005.87

QSR International Pty Ltd. (2018). *Nvivo* (Version 12 Plus) [Software]. https://lumivero.com/products/nvivo/

Raiden Network. (2019). *Fast, cheap, scalable token transfers for Ethereum.* https://github.com/raiden-network/raiden/

Raja, U., & Tretter, M. J. (2012). Defining and evaluating a measure of open source project survivability. *IEEE Transactions on Software Engineering*, *38*(1), 163–174. https://doi.org/10.1109/TSE.2011.39

Rapoport, A. (1983). The metaphor in the language of science. *Semiotische: Berlichte*, *12*(13), 25–43.

Rapport, D., Costanza, R., & McMichael, A. (1998). Assessing ecosystem health. *Trends in Ecology & Evolution*, *13*(10), 397–402. https://doi.org/10.1016/S0169-5347(98)01449-9

Rapport, D. J. (1989). What constitutes ecosystem health? *Perspectives in Biology and Medicine*, *33*(1), 120–132. https://doi.org/10.1353/pbm.1990.0004

Raymond, E. (1999). *The cathedral & the bazaar: Musings on linux and open source by an accidental revolutionary.* O'Reilly.

Renesse, R. V., & Altinbuken, D. (2015). Paxos made moderately complex. *ACM Computing Surveys*, *47*. https://doi.org/10.1145/2673577

Research Centre Holding Ltd. (2020, May 05). *Apograf publications on cryptography, distributed systems, blockchain.* https://apograf.io/

Rivest, R. L., & Shamir, A. (1996). PayWord and MicroMint: Two simple micropayment schemes. In *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)* (Vol. 1189, pp. 69–87). https://people.csail.mit.edu/rivest/RivestShamir-mpay.pdf

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Association for Computing Machinery*, *21*(2), 120–126.

Robinson, W. N., Deng, T., & Qi, Z. (2016). Developer behavior and sentiment from data mining open source repositories. *Proceedings of the Annual Hawaii International Conference on System Sciences*, 3729–3738. https://doi.org/10.1109/HICSS.2016.465

Robles, G., Amor, J. J., Gonzalez-Barahona, J. M., & Herraiz, I. (2005). Evolution and growth in large libre software projects. *International Workshop on Principles of Software Evolution (IWPSE), 2005*, 165–174. https://doi.org/10.1109/IWPSE.2005.17

Robles, G., & González-Barahona, J. M. (2012). A comprehensive study of software forks: Dates, reasons and outcomes. *Open Source Systems: Long-Term Sustainability*, 1–14.

Roldán, J. L., & Sánchez-Franco, M. J. (2012). Variance-based structural equation modeling: Guidelines for using partial least squares in Information Systems research. In *Research methodologies, innovations and philosophies in software systems engineering and Information Systems* (pp. 193–221). IGI Global. https://doi.org/10.4018/978-1-4666-0179-6.ch010

Romero, H. L., Dijkman, R. M., Grefen, P. W., Weele, A. J. V., & Jong, A. D. (2015). Measures of process harmonization. *Information and Software Technology*, *63*, 31–43. https://doi.org/10.1016/j.infsof.2015.03.004

Rosseel, Y. (2020). *Structural equation modeling with lavaan.* https://doi.org/10.1002/9781119579038

Rouibah, K., Lowry, P., & Al-Mutairi, L. (2015). Dimensions of business-to-consumer (B2C) systems success in Kuwait: Testing a modified DeLone and McLean IS success model in an e-commerce contex. *Journal of Global Information Management*, *23*, 41–70. https://doi.org/10.4018/JGIM.2015.07.0103

Russell, S. J., & Norvig, P. (2009). *Artificial intelligence: A modern approach* (3rd ed.). Prentice Hall.

Russo, D., & Stol, K. J. (2021). PLS-SEM for software engineering research: An introduction and survey. *ACM Computing Surveys*, *54*(4). https://doi.org/10.1145/3447580

Sai, A. R., Buckley, J., Fitzgerald, B., & Gear, A. L. (2021). Taxonomy of centralization in public blockchain systems: A systematic literature review. *Information Processing and Management*, *58*(4). https://doi.org/10.1016/j.ipm.2021.102584

Saini, M., Verma, R., Singh, A., & Chahal, K. (2020). Investigating diversity and impact of the popularity metrics for ranking software packages. *Journal of Software: Evolution and Process*, *32*(9). https://doi.org/10.1002/smr.2265

Satoshi Nakamoto Institute. (2023, May 05). *Bitcoin pre-release code.* https://satoshi.nakamotoinstitute.org/code/

Saunders, M., Lewis, P., & Thornhill, A. (2007). *Research methods for business students.* Financial Times; Prentice Hall.

Schaeffer, D. J., Herricks, E. E., & Kerster, H. W. (1988). Ecosystem health: I. measuring ecosystem health. *Environmental Management*, *12*(4), 445–455. https://doi.org/10.1007/BF01873258

Schilling, A. (2014). What do we know about floss developers' attraction, retention, and commitment? A literature review. *Proceedings of the Annual Hawaii International Conference on System Sciences*, 4003–4012. https://doi.org/10.1109/HICSS.2014.495

Schnorr, C. P. (1991). Efficient signature generation by smart cards. *Journal of Cryptology*, *4*(3), 161–174. https://doi.org/10.1007/BF00196725

Schroeder, M. (1993). A state-of-the-art distributed system: Computing with BOB. In *Distributed systems* (2nd ed., p. 1-16). ACM Press; Addison-Wesley. https://doi.org/10.5555/302430.302431

Schroer, J., & Hertel, G. (2009). Voluntary engagement in an open web-based encyclopedia: Wikipedians and why they do it. *Media Psychology*, *12*(1), 96–120. https://doi.org/10.1080/15213260802669466

Sein, M. K., Henfridsson, O., Purao, S., Rossi, M., & Lindgren, R. (2011). Action design research. *MIS Quarterly*, *35*(1), 37–56. https://doi.org/10.2307/23043488

Shaikh, M., & Levina, N. (2019). Selecting an open innovation community as an alliance partner: Looking for healthy communities and ecosystems. *Research Policy*, *48*(8), 103766. https://doi.org/10.1016/j.respol.2019.03.011

Shanks, G. (2002). Guidelines for conducting positivist case study research in Information Systems. *Australasian Journal of Information Systems*, *10*(1). https://doi.org/10.3127/ajis.v10i1.448

Sigalos, M. (2022, October 11). *The software used in bitcoin mining is getting its first big makeover in more than a decade—here's what's changing*. CNBC. https://www.cnbc.com/2022/10/11/bitcoin-mining-software-overhaul-stratum-v2-promoted-by-block-braiins.html

Six, N., Herbaut, N., & Salinesi, C. (2022). Blockchain software patterns for the design of decentralized applications: A systematic literature review. *Blockchain: Research and Applications*, *3*(2). https://doi.org/10.1016/j.bcra.2022.100061

Sjøberg, D. I. K., Dybå, T., Anda, B. C. D., & Hannay, J. E. (2008). Building theories in software engineering. In *Guide to advanced empirical software engineering* (pp. 312–336). Springer. https://doi.org/10.1007/978-1-84800-044-5_12

Skeen, D. (1981). Nonblocking commit protocols. *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, 133–142. https://doi.org/10.1145/582318.582339

Smirnova, I., Reitzig, M., & Alexy, O. (2022). What makes the right OSS contributor tick? Treatments to motivate high-skilled developers. *Research Policy*, *51*(1). https://doi.org/10.1016/j.respol.2021.104368

Smithson, S., & Hirschheim, R. (1998). Analysing Information Systems evaluation: Another look at an old problem. *European Journal of Information Systems*, *7*(3), 158–174. https://doi.org/10.1057/palgrave.ejis.3000304

Song, J. (2019, November 15). *On Altcoin Valuation.* Medium. https://medium.com/@jimmysong/on-altcoin-valuation-bf19a30ee0df

Spinellis, D. (2012). Git. *IEEE Software*, *29*(3), 100–101. https://doi.org/10.1109/MS.2012.61

Stallings, W. (2017). *Cryptography and network security, principles and practice.* Pearson.

Stallman, R. (2009). Why "open source" misses the point of free software. *Communications of the ACM*, *52*(6), 31–33. https://doi.org/10.1145/1516046.1516058

Strode, D. E. (2016). A dependency taxonomy for agile software development projects. *Information Systems Frontiers*, *18*(1), 23–46. https://doi.org/10.1007/s10796-015-9574-1

Sudhakar, G. P. (2012). A model of critical success factors for software projects. *Journal of Enterprise Information Management*, *25*(6), 537–558. https://doi.org/10.1108/17410391211272829

Sutton, R. I., & Staw, B. M. (1995). What theory is not. *Administrative Science Quarterly*, *40*(3), 371–384. https://www.jstor.org/stable/2393788

Szabo, N. (2008, December 27). *Bit gold.* blogspot. https://unenumerated.blogspot.com/2005/12/bit-gold.html

Szopinski, D., Schoormann, T., John, T., Knackstedt, R., & Kundisch, D. (2020). Software tools for business model innovation: Current state and future challenges. *Electronic Markets*, *30*, 469–494. https://doi.org/10.1007/s12525-018-0326-1

Szopinski, D., Schoormann, T., & Kundisch, D. (2019). Because your taxonomy is worth it : Towards a framework for taxonomy evaluation. *27th European Conference on Information Systems (ECIS)*. https://aisel.aisnet.org/ecis2019_rp/104

Tamburri, D. A., Palomba, F., Serebrenik, A., & Zaidman, A. (2019). Discovering community patterns in open-source: A systematic approach and its evaluation. *Empirical Software Engineering*, *24*(3), 1369–1417. https://doi.org/10.1007/s10664-018-9659-9

Tasca, P., & Tessone, C. (2019). Taxonomy of blockchain technologies: Principles of identification and classification. *Ledger*, *4*, 1–39. https://doi.org/10.5195/LEDGER.2019.140

Team Rocket. (2018). *Snowflake to Avalanche: A novel metastable consensus protocol family for cryptocurrencies.* https://ipfs.io/ipfs/QmUy4jh5mGNZvLkjies1RWM4YuvJh5o2FYopNPVYwrRVGV

The Apache Software Foundation. (2019, February 5). *Apache ZooKeeper.* https://zookeeper.apache.org/

The Free Software Foundation. (1989). *What is free software?* https://www.gnu.org/philosophy/free-sw.html

Thibault, L. T., Sarry, T., & Hafid, A. S. (2022). Blockchain scaling using rollups: A comprehensive survey. *IEEE Access*, *10*, 93039-93054. https://doi.org/10.1109/ACCESS.2022.3200051

Tönnissen, S., & Teuteberg, F. (2020). Analysing the impact of blockchain-technology for operations and supply chain management: An explanatory model drawn from multiple case studies. *International Journal of Information Management*, *52*. https://doi.org/10.1016/j.ijinfomgt.2019.05.009

Treiblmaier, H. (2018). The impact of the blockchain on the supply chain: A theory-based research framework and a call for action. *Supply Chain Management*, *23*(6), 545–559. https://doi.org/10.1108/SCM-01-2018-0029

Treiblmaier, H., & Sillaber, C. (2021). The impact of blockchain on e-commerce: A framework for salient research topics. *Electronic Commerce Research and Applications*, *48*. https://doi.org/10.1016/j.elerap.2021.101054

Tsamardinos, I., Brown, L. E., & Aliferis, C. F. (2006). The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, *65*(1), 31–78. https://doi.org/10.1007/s10994-006-6889-7

Vaishnavi, V., Kuechler, B., & Petter, S. (2004). *Design Science research in Information Systems.* http://www.desrist.org/design-research-in-information-systems/

van den Berk, I., Jansen, S., & Luinenburg, L. (2010). Software ecosystems: A software ecosystem strategy assessment model. *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, 127–134. https://doi.org/10.1145/1842752.1842781

Vasilescu, B., Blincoe, K., Xuan, Q., Casalnuovo, C., Damian, D., Devanbu, P., & Filkov, V. (2016). The sky is not the limit: Multitasking across GitHub projects. *ICSE '16: Proceedings of the 38th International Conference on Software Engineering*, 994–1005. https://doi.org/10.1145/2884781.2884875

Venable, J., Pries-Heje, J., & Baskerville, R. (2016). FEDS: A framework for evaluation in design science research. *European Journal of Information Systems*, *25*(1), 77–89. https://doi.org/10.1057/ejis.2014.36

Vijayasarathy, L., & Turk, D. (2012). Drivers of agile software development use: Dialectic interplay between benefits and hindrances. *Information and Software Technology*, *54*(2), 137–148. https://doi.org/10.1016/j.infsof.2011.08.003

VISA. (2019, April 25). https://www.visa.com

Vukolić, M. (2016). The quest for scalable blockchain fabric: Proof-of-work vs. BFT Replication. *Open Problems in Network Security*, 112–125.

Wahyudin, D., Mustofa, K., Schatten, A., Biffl, S., & Tjoa, A. M. (2007). Monitoring the "health" status of open source web-engineering projects. *International Journal of Web Information Systems*, *3*(1), 116–139. https://doi.org/10.1108/17440080710829252

Waldo, J., Wyant, G., Wollrath, A., & Kendall, S. (1994). *A note on distributed computing.* Sun Microsystems Laboratories Inc.

Walliman, N. (2011). *Research methods: The basics.* Routledge.

Wang, W., Hoang, D. T., Hu, P., Xiong, Z., Niyato, D., Wang, P., … Kim, D. I. (2019). A survey on consensus mechanisms and mining strategy management in blockchain networks. *IEEE Access*, *7*, 22328–22370. https://doi.org/10.1109/ACCESS.2019.2896108

Wang, Z., Feng, Y., Wang, Y., Jones, J. A., & Redmiles, D. (2020). Unveiling elite developers activities in open source projects. *ACM Transactions on Software Engineering and Methodology*, *29*(3). https://doi.org/10.1145/3387111

Wang, Z., & Perry, D. (2016). Role distribution and transformation in open source software project teams. *2015 Asia-Pacific Software Engineering Conference (APSEC)*, 119–126. https://doi.org/10.1109/APSEC.2015.12

Waterman, A., & Asanović, K. (Eds.). (2017). *The RISC-V instruction set manual, Volume I: user-level ISA, document version 2.2.* RISC-V Foundation.

Watkins, M. W. (2018). Exploratory factor analysis: A guide to best practice. *Journal of Black Psychology*, *44*(3), 219–246. https://doi.org/10.1177/0095798418771807

Weber, R. (2012). Evaluating and developing theories in the Information Systems discipline. *Journal of the Association for Information Systems*, *13*(1), 1–30. https://doi.org/10.17705/1jais.00284

Weick, K. E. (1995). What theory is not, theorizing is. *Administrative Sciences Quarterly*, *40*(3), 385–390. https://doi.org/10.2307/2393789

WeiDai. (1998). *Bmoney.* http://www.weidai.com/bmoney.txt

Weiss, D. (2005). Measuring success of open source projects using web search engines. *The First International Conference on Open Source Systems.* https://eprints.lincoln.ac.uk/id/eprint/76/

Wensley, J. H., Lamport, L., Goldberg, J., Green, M. W., Levitt, K. N., Melliar-Smith, P. M., … Weinstock, C. B. (1978). SIFT: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, *66*(10), 1240–1255. https://doi.org/10.1109/PROC.1978.11114

Wieninger, S., Schuh, G., & Fischer, V. (2019). Development of a blockchain taxonomy. *2019 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, 1–9. https://doi.org/10.1109/ice.2019.8792659

Willem E. Saris, A. S., & van der Veld, W. M. (2009). Testing structural equation models or detection of misspecifications? *Structural Equation Modeling: A Multidisciplinary Journal*, *16*(4), 561–582. https://doi.org/10.1080/10705510903203433

Wolf, E. J., Harrington, K. M., Clark, S. L., & Miller, M. W. (2013). Sample size requirements for structural equation models: An evaluation of power, bias, and solution propriety. *Educational and Psychological Measurement*, *73*(6), 913–934. https://doi.org/10.1177/0013164413495237

Wolfe, G. (1994, October 1). *The (second phase of the) revolution has begun.* Wired. https://www.wired.com/1994/10/mosaic/

Wood, G. (2014). *Ethereum: A secure decentralised generalised transaction ledger.* https://ethereum.github.io/yellowpaper/paper.pdf

Xiao, Y., Zhang, N., Lou, W., & Hou, Y. T. (2020). A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys and Tutorials*, *22*(2), 1432-1465. https://doi.org/10.1109/COMST.2020.2969706

Xu, X., Weber, I., Staples, M., Zhu, L., Bosch, J., Bass, L., … Rimba, P. (2017). A taxonomy of blockchain-based systems for architecture design. *Proceedings - 2017 IEEE International Conference on Software Architecture, ICSA 2017*, 243–252. https://doi.org/10.1109/ICSA.2017.33

Yakovenko, A. (2017). Solana: A new architecture for a high performance blockchain (version 0.8.13). https://solana.com/solana-whitepaper.pdf

Yamashita, K., Kamei, Y., McIntosh, S., Hassan, A., & Ubayashi, N. (2016). Magnet or sticky? Measuring project characteristics from the perspective of developer attraction and retention. *Journal of Information Processing*, *24*(2), 339–348. https://doi.org/10.2197/ipsjjip.24.339

Yuan, K.-H., & Bentler, P. M. (2000). Three likelihood-based methods for mean and covariance structure analysis with nonnormal missing data. *Sociological Methodology*, *30*(1), 165–200. https://doi.org/10.1111/0081-1750.00078

Zaleski, S., & Michalski, R. (2021). Success factors in sustainable management of IT service projects: Exploratory factor analysis. *Sustainability*, *13*(8). https://doi.org/10.3390/su13084457

Zamani, M., Movahedi, M., & Raykova, M. (2018). RapidChain: Scaling blockchain via full sharding. *Proceedings of the ACM Conference on Computer and Communications Security*, 931–948. https://doi.org/10.1145/3243734.3243853

Zuboff, S. (2019). *The age of surveillance capitalism: The fight for a human future at the new frontier of power.* PublicAffairs.

Zwirglmaier, K., Straub, D., & Groth, K. M. (2017). Capturing cognitive causal paths in human reliability analysis with Bayesian network models. *Reliability Engineering and System Safety*, *158*, 117–129. https://doi.org/10.1016/j.ress.2016.10.010

Appendix A

Mathematical Description of SEM

A.1 Model Definition

The mathematical notation is introduced presently. While not exhaustive, the mathematical notation can help understand what exactly it is the software is doing in an EFA, as well as to highlight commonalities and differences between statistical methods. The derivation in this section is primarily found in Finch (2020a), Hofacker (2007), and Harman (1976).

An EFA, or common factor model can be represented as follows using matrix notation:

$$\mathrm{Y} = \boldsymbol{\Lambda\eta} + \boldsymbol{\epsilon} \tag{A.1}$$

where,

- Y is the matrix of observed indicator variables. Each row in the matrix represents an individual project, and each column represents a different variable,
- $\boldsymbol{\Lambda}$ is the factor loading matrix. Each loading represents the effect of a factor on an observed variable. The loadings can be interpreted as the strength of the link between the factors and the observed variables,
- $\boldsymbol{\eta}$ is the matrix of factors. These are the underlying (latent) variables that are being identified. Each factor represents a possible cause of correlations among the observed variables, and
- $\boldsymbol{\epsilon}$ is the matrix of residuals. These are the parts of the observed variables that cannot be explained by the factors. Each error term is assumed to be uncorrelated with all the factors and all the other error terms.

Expanding Equation A.1 to show individual terms gives:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} \lambda_{11} & \lambda_{12} & \cdots & \lambda_{1m} \\ \lambda_{21} & \lambda_{22} & \cdots & \lambda_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{n1} & \lambda_{n2} & \cdots & \lambda_{nm} \end{bmatrix} \begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix} \tag{A.2}$$

where,

- y_i are the observed variables,
- η_j are the latent factors,
- λ_{ij} are the factor loadings, which represent the effect of the jth factor on the ith observed variable, and
- ϵ_i are the unique errors associated with each observed variable.

Correlation and Covariance Matrices

A note on the correlation and covariance matrices. A correlation matrix has the diagonals set to unity as an item perfectly correlates with itself; also the same as the covariance matrix of z (standardised) scores, shown here with three indicator variables as a 3×3 matrix:

$$\mathbf{R} = \begin{bmatrix} 1 & r_{12} & r_{13} \\ r_{21} & 1 & r_{23} \\ r_{31} & r_{32} & 1 \end{bmatrix} \tag{A.3}$$

Generally a correlation matrix is used to estimate the model in EFA because of the standardised nature. Loadings are expected to be between -1 and 1. A loading of 1 means that the observed variable is perfectly positively correlated with the factor, a loading of -1 means that it's perfectly negatively correlated. By default, the `fa()` function from the `psych` package uses the correlation matrix of the provided data. This is because, in many contexts, it's more appropriate to use the correlation matrix for factor analysis, as it standardises the variables, removing the influence of differing scales.

This is in contrast to the default behaviour for CFA and SEM with `lavaan` which fit the model based on a covariance matrix. Most software allows for either correlation or covariance matrix as input data and the user can specify what is needed. The present work is fitting a model to a raw dataset, so both `psych` and `lavaan` calculate the necessary matrix according to their default procedure.

A covariance (or variance-covariance) matrix has the variance in the diagonal (covariance of each element with itself) and covariances off the diagonal:

$$\mathbf{S} = \begin{bmatrix} \sigma_{11}^2 & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22}^2 & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33}^2 \end{bmatrix} \tag{A.4}$$

This is non-standardised and the variance is represented in terms of the real units of input. For example, a variance of $\sigma_{x_i} = 98.5$ where x is commits on a three-month basis means 98.5 commits (squared) of variance. The off-diagonal elements of the matrix (for example, σ_{12}, σ_{13}, σ_{23}) represent the covariances between pairs of variables. These covariances are a measure of how much two variables change together.

The correlation between x_1 and x_2 is the same as the correlation between x_2 and x_1 (these are double-headed arrows in Figure 4.4) and so the matrix $\mathbf{R}$ (and also S) is symmetric and can be represented without the upper triangular portion:

$$\mathbf{R} = \begin{bmatrix} 1 & & \\ r_{21} & 1 & \\ r_{31} & r_{32} & 1 \end{bmatrix} \tag{A.5}$$

Using the factor model in Equation A.1, an estimated (model-implied) covariance matrix $\mathbf{\Sigma}$ can be calculated according to:

$$\mathbf{\Sigma} = \mathbf{\Lambda \Psi \Lambda'} + \mathbf{\Theta} \tag{A.6}$$

where:

- $\mathbf{\Sigma}$ = predicted covariance matrix of the indicators according to the model,
- $\mathbf{\Lambda}$ = the factor loading matrix,
- $\mathbf{\Psi}$ = factor correlation matrix, and
- $\mathbf{\Theta}$ = unique error variances.

This estimated model-implied covariance matrix $\mathbf{\Sigma}$ is different from the earlier correlation matrix $\mathbf{R}$. $\mathbf{R}$ is obtained using the data sample and is intended to represent the population whereas $\mathbf{\Sigma}$ is from the model. Specifically, the different between these two is what EFA (and SEM) are assessing. If the model matches the data well, then there will be little difference between $\mathbf{\Sigma}$ and $\mathbf{R}$. The farther off the correlation matrices are, the less the model is considered a good approximation of the data.

A.1.1 Eigenvalues

An eigenvalue is a scalar associated with a linear system of equations (often represented as a matrix), which arises from a characteristic equation of the system. For a square matrix $\mathbf{A}$ and a vector $\mathbf{v}$, an eigenvalue λ satisfies the following equation: $\mathbf{Av} = \lambda\mathbf{v}$ where $\mathbf{v}$ is an eigenvector corresponding to the eigenvalue λ.[1] In the context of factor analysis, an eigenvalue represents the variance in all the variables which is accounted for by that single latent factor. Higher eigenvalues correspond to factors that explain a greater proportion of the variance in the data, as is seen shortly.

1. Derivation and eigenvalue mathematics is left out as is a common operation handled by statistical software. More analytical details can be found in Harman (1976).

In factor analysis, the resultant eigenvalues of the correlation matrix $\mathbf{R}$ can be interpreted as the amount of variance in the observed variables that is accounted for by each factor. A factor with a larger eigenvalue accounts for a larger amount of the total variance. This is important when deciding how many factors to retain in an exploratory factor analysis, as factors with small eigenvalues contribute little to the explanation of variances in the variables and can be dropped from further considerations.

A.1.2 Relation to Multiple Linear Regression

The EFA equation (A.1) is structurally similar to multiple linear regression. Both models aim to explain a dependent variable through a combination of other variables. In multiple linear regression, the observed variables are expressed as a linear combination of predictor variables and an error term.

In the context of a regression analysis, this corresponds to the linear model equation

$$\mathrm{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{A.7}$$

where:

- Y is a vector of observations for the dependent variable,
- $\mathbf{X}$ is a matrix where each column is a vector of observations for a particular independent variable,
- $\boldsymbol{\beta}$ are the regression coefficients representing the change in the dependent variable per unit change in the respective independent variable, and
- $\boldsymbol{\epsilon}$ is a vector of the error terms.

Similarly, in EFA, the observed variables are modelled as a linear combination of latent factors and unique error. The main difference lies in the nature of the explanatory variables: In regression, these are observed variables, while in EFA, these are latent variables, which, by definition can not be observed.

Linear regression is a straightforward statistical technique that predicts a single dependent variable from one or more independent variables. On the other hand, SEM is a multivariate statistical framework that allows for simultaneous estimation of multiple regressions, modelling of latent variables, and testing of causal relationships. SEM extends regression by allowing for latent variables (factors), measurement error, and models involving multiple dependent variables.

Appendix B
Database Queries

B.1 Introduction

A database is built of all GitHub activity beginning with `JSON` from the GitHub Archive; details are in Section 6.3.3. The database is queried from the list of blockchain and cryptocurrency repositories from CoinMarketCap's top 600 projects by market capitalisation and manually cleaned and verified.

The SQL queries are used to calculate metrics. Each query is a concatenated string based on a repository location in the database, for example, `'bitcoin/bitcoin'`. The main script to process the queries is shown in Appendix B.2. Other scripts to gather and process data are not included, but can be found in the GitHub repository.[1]

1. https://github.com/millecodex/phd

B.2 Python Query Script

```
# ------------------------------------------------------
# -- R U N   Q U E R Y ---------------------------------
# ------------------------------------------------------
def runQuery(column_name, df, query_L, query_R=''):
    """Executes a Clickhouse query on a Pandas dataframe;
    the complete query is a string 'query_L + repo + query_R' where
    repo is the name of a GitHub repository contained in df

    Keyword arguments:
    column_name -- the new column to be added to the dataframe, e.g. 'stars'
    df -- the pandas dataframe containing 'repo' and 'forge' columns
    query_L -- the SQL string preceding the GitHub repository name
    query_R -- the SQL string postceding the GitHub repository name, this can be empty by
     ↪  default
    """
    start = datetime.now()
    num = 0
    num_u = 0
    for row in df.itertuples():
        # only GitHub for now as the client is connected to github_events DB
        if row.forge == 'github':
            repo = row.repo
            # skip the NaN repos
            if type(repo) == str:
                query = query_L + '\'' + repo + '\'' + query_R
                result = client.execute(query)
                num += 1

              # query returns a tuple of list elements accessible by [first list][first item]
                # empty list returns -1, meaning it has to be manually verified
                # average of zero returns a NaN
                if len(result) == 0:
                    df.at[row.Index, column_name] = -1
                elif math.isnan(result[0][0]):
                    df.at[row.Index, column_name] = 0
                else:
                    df.at[row.Index, column_name] = result[0][0]
                num_u += 1

                # proof-of-life
                if num % 10 == 0:
                    sys.stdout.write(".")
                    sys.stdout.flush()

    # some log info
    now = datetime.now()
    elapsed = (now - start).total_seconds()
    output = column_name + ': ' + str(num) + ' repos queried and ' + str(num_u) + ' updated;
     ↪  Query took ' + str(round(elapsed, 2)) + ' seconds.'
    print('\n' + output)
```

B.3 SQL Queries

```
/*
# ------------------------------------------------
# ----S T A R S ----------------------------------
# ------------------------------------------------
*/
SELECT count()
FROM github_events
WHERE event_type = 'WatchEvent' AND repo_name = 'bitcoin/bitcoin';
```

Forks

```
/*
# ------------------------------------------------
# ----F O R K S ----------------------------------
# ------------------------------------------------
*/
SELECT count()
FROM github_events
WHERE (event_type = 'ForkEvent') AND repo_name = 'bitcoin/bitcoin'
```

```
/*
# ------------------------------------------------------
# --- A U T H O R S ------------------------------------
# ------------------------------------------------------
# Calculates a monthly average from previous 3 months
# excluding current month because it is in progress
*/
SELECT
    ROUND(SUM(authors) / COUNT(month), 2) AS average
FROM
    (
        SELECT
            uniq(actor_login) AS authors,
            toMonth(created_at) AS month,
            toYear(created_at) AS year
        FROM
            github_events
        WHERE
            event_type IN ('PullRequestEvent', 'IssuesEvent', 'IssueCommentEvent',
             ↪  'PullRequestReviewCommentEvent')
            AND repo_name = 'bitcoin/bitcoin'
            AND created_at >= dateSub(MONTH, 3, toStartOfMonth(now()))
            AND created_at < toStartOfMonth(now())
        GROUP BY
            month, year
        ORDER BY
            year DESC, month DESC
    )
```

```
/*
# ------------------------------------------------------
# --- C O M M I T S ------------------------------------
# ------------------------------------------------------
# Calculates a monthly average from previous 3 months
# excluding current month because it is in progress
#
# note: there will be moderate timezone discrepancies,
#       especially when calculating near the first of the
#       month
*/
SELECT
    ROUND( SUM(sum_push_distinct) / COUNT(month), 2) AS average
FROM
    (
        SELECT
            SUM(push_distinct_size) AS sum_push_distinct,
            toMonth(created_at) AS month,
            toYear(created_at) AS year
        FROM
            github_events
        WHERE
            repo_name = 'bitcoin/bitcoin'
            AND event_type = 'PushEvent'
            AND created_at >= dateSub(MONTH, 3, toStartOfMonth(now()))
            AND created_at < toStartOfMonth(now())
        GROUP BY
            month, year
        ORDER BY
            year DESC,
            month DESC
    )
```

```
/*
# ------------------------------------------------------
# --- C O M M E N T S ----------------------------------
# ------------------------------------------------------
# Calculates a monthly average from previous 3 months
# excluding current month because it is in progress
#
# total COMMENTS includes all commenting activity
# any comments counts as activity and increase engagement
# there are 3 event_type comment events:
# > CommitCommentEvent
# > IssueCommentEvent
# > CommitCommentEvent
*/
SELECT
    ROUND(SUM(total) / COUNT(month), 2) AS average
FROM
    (
        SELECT
            (
                uniqIf(comment_id, event_type = 'PullRequestReviewCommentEvent') +
                uniqIf(comment_id, event_type = 'IssueCommentEvent') +
                uniqIf(comment_id, event_type = 'CommitCommentEvent')
            ) AS total,
            toMonth(created_at) AS month,
            toYear(created_at) AS year
        FROM
            github_events
        WHERE
            repo_name = 'bitcoin/bitcoin'
            AND created_at >= dateSub(MONTH, 3, toStartOfMonth(now()))
            AND created_at < toStartOfMonth(now())
        GROUP BY
            month,
            year
        ORDER BY
            year DESC,
            month DESC
    )
```

```
/*
# ------------------------------------------------------
# --- P U L L   R E Q U E S T S   O P E N E D ----------
# ------------------------------------------------------
# Calculates a monthly average from previous 3 months
# excluding current month because it is in progress
*/
SELECT
    ROUND(SUM(opened) / COUNT(month), 2) AS average
FROM
    (
        SELECT
            SUM(action = 'opened') AS opened,
            toYear(created_at) AS year,
            toMonth(created_at) AS month
        FROM
            github_events
        WHERE
            repo_name = 'bitcoin/bitcoin'
            AND event_type = 'PullRequestEvent'
            AND created_at >= dateSub(MONTH, 3, toStartOfMonth(now()))
            AND created_at < toStartOfMonth(now())
        GROUP BY
            month,
            year
        ORDER BY
            year DESC,
            month DESC
    )
```

```
/*
# ------------------------------------------------------
# --- A V E R A G E  C A L C U L A T I O N ---------------
# ------------------------------------------------------
*/
WITH
    repo AS 'bitcoin/bitcoin',
    total_days AS (
        SUM(dateDiff('minute', toDateTime(opened), toDateTime(closed)))/60/24
    ),
    mins_open AS (
        ROUND(dateDiff('second', toDateTime(opened), toDateTime(closed)), 2)/60
    ),
    num_issues AS (
        COUNT()
    ),
    event AS (
        (event_type = 'IssuesEvent' OR event_type = 'PullRequestEvent')
    ),
    created AS (
        created_at >= toDateTime('2019-11-01')
    )
SELECT
    ROUND(total_days / num_issues, 2) AS average_response_time_days
FROM
    (
        SELECT *
        FROM
            (
                SELECT
                    number,
                    created_at AS opened
                FROM
                    github_events
                WHERE
                    repo
                    AND event
                    AND action = 'opened'
                    AND created
            ) AS t1
        INNER JOIN
            (
                SELECT
                    number,
                    created_at AS closed
                FROM
                    github_events
                WHERE
                    repo
                    AND event
                    AND action = 'closed'
                    AND created
            ) AS t2 USING (number)
    )
WHERE
    mins_open > 5
```

```
/*
# ------------------------------------------------------
# --- M E D I A N   C A L C U L A T I O N ---------------
# ------------------------------------------------------
*/
WITH
    repo AS 'bitcoin/bitcoin',
    mins_open AS (
       ROUND(dateDiff('second', toDateTime(opened), toDateTime(closed)), 2) / 60 ),
    days_open AS (
        ROUND(dateDiff('second', toDateTime(opened), toDateTime(closed)), 2) / 60 /
         ↪  60 / 24 ),
    event AS (
        (event_type = 'IssuesEvent' OR event_type = 'PullRequestEvent')),
    created AS (
        created_at >= dateSub('year', 1, now())
    )
SELECT
    ROUND(medianDeterministic(days_open, 1), 2) AS median_response_time_days
FROM
    (
        SELECT *
        FROM
            (
                SELECT
                    number,
                    created_at AS opened
                FROM
                    github_events
                WHERE
                    repo
                    AND event
                    AND action = 'opened'
                    AND created
            ) AS t1
        INNER JOIN
            (
                SELECT
                    number,
                    created_at AS closed
                FROM
                    github_events
                WHERE
                    repo
                    AND event
                    AND action = 'closed'
                    AND created
            ) AS t2 USING (number)
    )
WHERE
    mins_open > 5
```

```sql
/*
# ------------------------------------------------------
# --- L O N G E V I T Y  C A L C U L A T I O N ---------
# ------------------------------------------------------
# searches the repo for all activity by a contributor
# excludes those that step in 'one-time'
# such as starring or forking or leaving a comment and never returning
# this excludes devs that work on other projects
# (only calculates from single repo)
#------------------------------------------------------
# view multiple author's days active
#------------------------------------------------------
*/
WITH
    days_active AS (
        dateDiff('day', toDateTime(earliest_seen), toDateTime(last_seen))
    )
SELECT
    days_active,
    actor_login,
    last_seen
FROM
    (
        SELECT
            MIN(created_at) AS earliest_seen,
            MAX(created_at) AS last_seen,
            days_active,
            actor_login
        FROM
            github_events
        WHERE
            repo_name = 'bitcoin/bitcoin'
        GROUP BY
            actor_login
        ORDER BY
            days_active DESC
    )
WHERE
    days_active > 0
LIMIT
    100
```

```
/*
# ------------------------------------------------------
# --- A V G.  D E V.  L O N G E V I T Y ----------------
# ------------------------------------------------------
*/
WITH
    days_active AS (
        dateDiff('day', toDateTime(earliest_seen), toDateTime(last_seen))
    )
SELECT
    ROUND((SUM(days_active) / COUNT()), 2) AS avg_dev_days_active
FROM
    (
        SELECT
            MIN(created_at) AS earliest_seen,
            MAX(created_at) AS last_seen,
            days_active,
            actor_login,
            COUNT()
        FROM
            github_events
        WHERE
            repo_name = 'bitcoin/bitcoin'
        GROUP BY
            actor_login
        ORDER BY
            days_active DESC
    )
WHERE
    days_active > 0
```

A Note on the Type

The main type is set in **Linux Libertine**, a digital typeface designed by Philipp H. Poll at Libertine Open Fonts Project foundry, which aims to create free and open alternatives to proprietary typefaces such as Times New Roman.

Tables, Figures, and Captions are set in **Linux Biolinum**, the sans-serif companion font by the same designer and foundry.

They are developed with the free font editor FontForge and are licensed under the GNU General Public License and the SIL Open Font License. See more at https://en.wikipedia.org/wiki/Linux_Libertine.

`Titles, headings, code, and \texttt environments are set in` **`Ubuntu Mono`** `of the Ubuntu Font Family which debuted in Ubuntu 10.10 in 2010.` It is designed by the Dalton Maag Font Foundry founded by Swiss typographer Bruno Maag in 1991. The Truetype and OpenType design files are distributed under the SIL Open Font License to encourage modification and dissemination. The typeface is a Humanist sans-serif, uses OpenType features and is manually hinted for clarity on desktop and mobile computing screens. See more at https://en.wikipedia.org/wiki/Ubuntu_(typeface).

A Note on Software

Software used in the ideation, research, and production of this thesis includes the following. Where possible, free and open source software (FOSS) is chosen as the primary product. Auckland University of Technology (AUT) holds a Microsoft 365 Education license that provides students with access to Microsoft products, and an Adobe for Enterprise license.

Software	Application	License	Licensee
Acrobat Pro	PDFLaTeX & XeTeX output and annotations	Adobe for Enterprise	AUT
Brave	Web browser	MPL[1] 2.0	FOSS
ChatGPT Plus	Grammar and spelling; debugging	Proprietary (OpenAI)	Personal*
ClickHouse	Database management system	Apache License 2.0	FOSS
JupyterLab	Scripting; testing	MIT	FOSS
macOS	Operating system	Proprietary (Apple)	Personal*
Mendeley	Reference management; document annotation	Proprietary (Elsevier)	Personal
Microsoft 365	Outlook, PowerPoint, Excel, Word, Teams, and OneDrive	Microsoft 365 Education	AUT
Notepad++	Text editing	GNU GPL v3	FOSS
NVivo	Textual analysis	Proprietary (QSR Int'l)	AUT
Overleaf	LaTeX cloud editor	AGPL v3	FOSS
Python†	Scripting	PSFL[2] 2	FOSS
R†	Statistical analysis	GNU GPL v2	FOSS
RStudio	Graphical user interface	AGPL v3	FOSS
TeX	Typesetting	Permissive License	FOSS
TeXstudio	PDFLaTeX & XeTeX thesis creation	GPL 2.0	FOSS
Ubuntu	Operating system	GNU GPL[3]	FOSS
VS Code	Scripting; text editing	MIT	FOSS
Windows	Operating system	Proprietary (Microsoft)	AUT
Zoom	Video conferencing	Proprietary (Zoom)	Personal

† Individual Python and **R** packages are referenced in the text; see pp.116,127,128,134,153,161,176.

* Paid version

1 Mozilla Public License

2 Python Software Foundation License

3 Ubuntu is a collection of components with varying licenses, see https://ubuntu.com/legal/open-source-licences

Data Anchoring

Data Anchoring, or Digital Archiving, or Public Timestamping (Haber & Stornetta, 1991), involves writing a cryptographic hash to immutable storage. Public blockchains present an ideal use case for this type of activity as the distributed database ensures data replication and censorship resistance, and the public nature makes verification quick and efficient.

The work has been anchored to popular blockchains by including the hash in a transaction in the appropriate style.

The `SHA256` hash of version 1 of this document is:[2]

```
9885dee02e36e2e4bbf808093364f2dafcc284a119f280417025e64fb5b47215
```

and can be verified by downloading the pdf file from GitHub[3], computing the SHA256 hash, for example: `$ shasum -a 256 Nijsse-PhD-anchored.pdf`, and comparing the hashed output to the public value found in the blockchain.[4]

Bitcoin

The `OP_RETURN` script opcode in a Bitcoin transaction marks the transaction outputs as unspendable, or specifically as a UTXO of type `nulldata`. Thus, any data in the field terminates the UTXO chain and can be used to burn bitcoin, or, as in this case, store 80 bytes arbitrary data. Using `OP_RETURN` is considered more polite than writing data to the pay-to-public-key-hash (p2pkh) output, as it is difficult to distinguish from a real public-key-hash, and so must be stored by all nodes (Bartoletti & Pompianu, 2017). `OP_RETURN` data can optionally be pruned by nodes to save storage.

2. You are likely reading version 2, available at https://tuwhera.aut.ac.nz/
3. https://github.com/millecodex/phd/blob/main/thesis/Nijsse-PhD-anchored.pdf
4. Any update to a pdf, such as including a transaction hash or block number, will invalidate the hash, so to avoid this issue, a live-update must be provided, exclusive of the original hash creation.

The transaction[5] in block `838705` contains the following transaction output, partially shown in `JSON`, where part of the `OP_RETURN` is:


```
"outputs":
[
  {
    "address":   null,
    "pkscript": "6a209885dee02e36e2e4bbf808093364f2dafcc284a119f280417025e64fb5b47215",
    "value":     0,
    "spent":     false,
    "spender":   null
  }
```


The end of the `pkscript` value matches the `SHA256` output.

Ethereum

Smart contract functionality allows for more versatile storage in Ethereum. One method is to deploy a contract that contains some data that can be retrieved later by calling the contract. An alternate method is to use the logging capability in the `data` field of a transaction that will write log data to the transaction receipt.

A contract is deployed to Ethereum at: `0x975A313f03a56d232D9353830D8930481CFf5e1f` which accepts a 32 byte hash and a message:

```
addHash(bytes32 _hash, string calldata _message)
```

In block 19666573, the transaction[6] contains the hash above (prepended with `0x`) and the message: 'Nijsse-PhD-data-anchor'.

The input data that formed the transaction contains our hash, highlighted here.


```
0x31d549f89885dee02e36e2e4bbf808093364f2dafcc284a119f280417025e64fb5b4721500
0000000000000000000000000000000000000000000000000000000000000040000000000000000
0000000000000000000000000000000000000000000000000164e696a7373652d5068442d6461
74612d616e63686f7200000000000000000000
```


5. Bitcoin Transaction ID: `76adf298dcd1d981fde6a846956ea86ad42cc26f8caba09c1ced920c943c4e04`
6. Ethereum Transaction ID: `0x847a818aec4827fc91260c6d1c3d85f53bb028381c48c2457c1bdfb82d5c8f93`