Runtime Evolution of Bitcoin's Consensus Rules

Jakob Svennevik Notland [1], Mariusz Nowostawski [2], and Jingyue Li [2]

[1]Norwegain University of Science and Technology
[2]Affiliation not available

October 30, 2023

Runtime Evolution of Bitcoin's Consensus Rules

Jakob S. Notland, Mariusz Nowostawski, and Jingyue Li

Abstract—Runtime evolution of a system concerns the ability to make changes during runtime without disrupting the service. Blockchain systems need to provide continuous service and integrity. Similar challenges have been observed in centrally controlled distributed systems that handle runtime evolution mainly by supporting compatible changes or running different versions concurrently. However, these solutions are not applicable in the case of blockchains, and new solutions are required. This study investigates over a decade of Bitcoin consensus evolution through their development channels using Strauss'grounded theory approach and root cause analysis. The results show nine deployment features which form nine deployment techniques and ten lessons learned from Bitcoin.

Index Terms—Bitcoin, blockchain, consensus, grounded theory, root cause analysis, runtime evolution

1 Introduction

Deployment of consensus changes is one of the most controversial [1], [2] and error-prone [3], [4], [5] activities in a blockchain. These changes redefine the fundamental behaviour in a blockchain, which can affect its security and the value of its currency. Trivial change could cause disruption, which could result in suspended services [6], lost mining revenue [4] and theft [7], [8].

This study considers the longest-living blockchain project: Bitcoin. The motivation of this work is to collect knowledge on evolutionary techniques in this specific blockchain and to gather *unknown known* security requirements [9]. The proposed techniques and security requirements may be well known to a seasoned Bitcoin developer. However, any other blockchain engineer may have to shuffle through thousands of unstructured data samples before acquiring the same knowledge. This research was conducted to understand the implications of consensus rule changes and techniques for a safe transition. The research questions are divided in two, considering the current practice for consensus changes in blockchain and the lessons learned.

- RQ1: What techniques have been applied to deploy consensus changes?
- RQ2: What are the lessons learned from deploying consensus changes?

This paper is the first to consider blockchain and consensus change at runtime. The study has been conducted as a qualitative analysis, covering 34 consensus rule changes over more than a decade of Bitcoin development (Appendix A, figure 7), and entails 1700 samples (Appendix B, figure 8 and 9). Samples correspond to email threads, forum threads, Github issues/pulls, IRC days, proposals, and others. The results overview techniques and best practices for consensus rule changes. The further evaluation shows how different factors impact secure deployment.

- *J.S. Notland and J. Li are with the Department of Computer science, Norwegian University of Science and Technology, Trondheim, Norway. E-mail: jakob.notland@ntnu.no, jingyue.li@ntnu.no*
- *M. Nowostawski are with the Department of Computer science, Norwegian University of Science and Technology, Gjøvik, Norway. E-mail: mariusz.nowostawski@ntnu.no*

Manuscript received April 19, 2005; revised August 26, 2015.

The remaining structure is as follows: Section 2 shows the background on blockchain and describes the nature of software and system evolution. Section 3 describes the application of grounded theory (GT) and root cause analysis. Section 4 presents the results, describing deployment techniques for rule changes in Bitcoin and the root causes for failure. Section 5 discusses the results before the conclusion in Section 6. Future work is elaborated in Section 7.

2 Background

2.1 The Bitcoin consensus protocol

Bitcoin is "a peer-to-peer electronic cash system" [10] consisting of a chain of blocks containing transaction history, as illustrated by figure 1. Any new block must abide by the consensus rules to be regarded as valid by the nodes in the network. Miners attempt different values for the nonce variable in a brute-force manner to produce a SHA-256 hash based on the entire block. The miners find a valid nonce for the block header when the resulting hash meets the required target difficulty. The difficulty indicates that the resulting hash must have a certain number of leading zeros. This mechanism is known as Proof-of-Work (PoW), first proposed to prevent email spam [11] and later applied for cryptocurrencies [12]. In the case of Bitcoin, PoW prevents Sybil attacks [13] and provides an immutability feature. The strength of these principles is preserved by the difficulty adjustment algorithm [10], ensuring that the network will produce blocks with an average rate of around ten minutes. Miners must comply with further consensus rules. Generally, blocks and transactions must be in a valid format. Miners are incentivised to follow these rules by collecting fees and the block reward generated in a special coinbase transaction with no inputs from previous transactions.

Block collisions occur when two new valid blocks are produced at the same height at approximately the same time. Collisions are resolved by *the longest (valid) chain rule* as specified in Nakamoto's whitepaper: "The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it" [10]. The notion of a *valid* chain is important because validness is subjective from the implementation's point of view. Any collision should

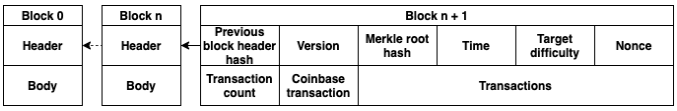

Fig. 1. Bitcoin blocks in a blockchain showing the content of the header and the body.

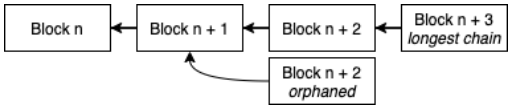

Fig. 2. Choosing the longest chain in the event of a collision.

quickly resolve when another block is appended on top of one of the colliding blocks. This behaviour implies that blocks have a slight chance of being *orphaned* (discarded) as illustrated in figure 2. The rule of thumb to prevent financial loss from orphaned blocks is to wait until the block containing the relevant transaction has at least six blocks built on top [14]. Collisions happen naturally, by an attack [15], or by inconsistent consensus validation [4].

2.2 Bitcoin consensus evolution

An inherent ideology within Bitcoin and especially Bitcoin Core (BTC) indicates what kind of changes are viable or controversial. One quote from the creator(s) can be seen as a cornerstone of this ideology.

```
"The nature of Bitcoin is such that once version
0.1 was released, the core design was set in stone
for the rest of its lifetime."
```

Listing 1. Satoshi 2010-06-17 Forum, ID: 195

The statement from Nakamoto explains that the system itself, as well as the original specification [10] defines Bitcoin's fundamental behaviour. Moreover, it highlights the importance of non-disruptive changes, no matter how insignificant they seem. This paper distinguishes between Bitcoin in general, BTC and Bitcoin Cash (BCH), who have different approaches to consensus changes. Consensus rule changes in BTC should preferably allow backward-compatibility such that legacy nodes can accept any new behaviour, keeping the network consistent (so-called soft fork). Backwards-incompatible changes (so-called hard forks) are prefered in BCH and sometimes required if the fundamental implementation does not work as intended. Such a change could be essential to prevent exploits or allow for the adaption and survival of the system. The Bitcoin community is sceptical of consensus changes and making them a habit because bugged or ill-intended code may be deployed and disrupt the integrity and stability of the system.

This paper will refer to conflicting blocks as *chain splits*. Chain splits can be *temporary*; less than six blocks will be orphaned, *persistent*; six or more blocks are orphaned, or *permanent*; both chains are expanded independently for all foreseeable future. An accidental chain split caused by inconsistent validation among the nodes will be referred to as a *consensus failure*. The longest chain rule is the most fundamental factor in deciding whether a rule change is successfully adopted in Bitcoin. The longest chain rule implies that the majority of miners can apply a network-wide backwards-compatible rule change. Relying on the majority is the prefered technique to deploy changes in BTC.

In Bitcoin, one may rely on the majority (>50%) of block-producing nodes (n) to perform a backwards-compatible consensus change. The reason for this to work is because the Nakamoto consensus model has a Byzantine Fault Tolerance (BFT) [16] threshold of 50% (f). Other consensus models might have different fault tolerance, such as 33% or 20% [17]. This paper uses the term *super-majority* (SM) to generalize the minimal threshold requirement for different deployment techniques and to describe the required threshold where faulty nodes are equal to or less than the tolerated threshold. A super-majority of abiding nodes (h) is denoted as $h>(n-f)$.

Consensus changes in BTC's history have mainly been deployed with techniques such as IsSuperMajority (ISM) [18], and the more established miner-activated soft fork (MASF) [19]. Other well-known techniques are the user-activated soft fork (UASF) [20] or BCH's prefered user-activated hard fork (UAHF) [21].

2.3 Software evolution

Software evolution is a field entailing processes and models for changing software. Within this domain, there is the subfield of runtime evolution [22]. Relevant to blockchain and this study is mainly the challenge of avoiding service outages while changing the running system. In the case of blockchain, this also relies on consistency in the network. Combined with the strict requirements of partition tolerance, there is a challenge of minimizing the impact of a change on the consistency, availability, and partition tolerance of the system (CAP) [23].

2.4 Evolution and maintenance in distributed systems

Before the decentralised networks, there were, and still are, distributed networks dominating sectors that blockchains have been envisaged to handle, such as finance [24] [25], logistics [26] [27], and healthcare [28] [29]. Researchers in this area seem to have ended up with satisfactory frameworks for maintenance where updates are deployed with central control and rely on either being compatible [30] or running different versions in parallel [31]. In these cases, changes have been deployed with techniques such as fast reboot, rolling upgrade, and big flip [32]. However, we argue that these methods are not directly applicable when deploying consensus rule changes on a blockchain. First, known techniques are hard to coordinate without central administration. Second, a consensus rule change in a blockchain conflicts with legacy rules as it changes the set of valid actions. Some techniques may even drastically affect the stability of a blockchain, such as the total hash power in the network. Thirdly, running different versions in parallel is problematic because it may lead to consensus failure.

3 Research Design

3.1 Research motivation and research questions

Blockchains must evolve to meet current and future requirements. Although there has been researched on different kinds of consensus code changes in blockchain [33], there is still a lack of understanding of *how* these changes are

deployed. That is crucial because failed deployments can severely affect a blockchain network. The consequences can result in financial loss for any invested participant or loss of faith in the system. Therefore this study investigated how consensus changes can be deployed (RQ1) and what lessons can be learned from past failures (RQ2).

3.2 Research method

The sheer amount of available samples on the world-wide-web would allow for quantitative and qualitative analysis. The authors chose to use a qualitative and inductive approach to achieve an insightful and holistic view of consensus changes in blockchains. Strauss' approach of grounded theory (GT) has been chosen as it is more applicable to studies with predefined research questions [34]. The GT approach is an iterative and recursive approach where the researchers must go back and forth until they achieve theoretical saturation. That is, when new samples stop contributing to the developing theories. The observations done throughout the study are covert [35], where a researcher can get the most authentic experience of how the actors conduct a process. Although covert observations can be ethically questionable, it is crucial to consider that the public archives of Bitcoin were created for this purpose and to provide transparency and accountability.

3.3 Research implementation

With the scope in mind, the study started with purposive sampling [35] of data, specifically from Bitcoin Core's development channels. Samples were discovered in these archives by filtering and purposively selecting (*cherry-picking*) samples relevant to consensus rule changes. The selected samples are efficiently sorted through the use of flexible coding [36]. Further, snowball sampling and triangulation [37] to avoid limitations from the initial samples.

Root cause analysis has been utilized to address lessons learned (RQ2) by drawing an Ishikawa diagram [38]. The approach is similar to "Discovering unknown known security requirements" [9] as it also uses concepts from GT and root cause analysis with incident fault trees.

A graph of the research method implemented is outlined in figure 3. The oval shapes indicate processes, the lines show the process flow, the rectangular shapes indicate data objects, and the cylinders indicate archives. Different colours highlight whether the concepts relate to data sampling, GT analysis, or root cause analysis. Appendix B further describe the method processes.

4 Results

The analysis of evolution in Bitcoin reveals the system, actors, and their processes to develop in a symbiosis of chaos tamed by the underlying infrastructure. It was challenging to identify the causalities in the decentralised environment. However, as the analysis commenced with codes giving meaning to the data, data triangulation revealed the relationships within. Eventually, this formed a chain of events, shown in the timeline in Appendix A, figure 7.

Initially, the categories discovered and applied through grounded theory are introduced in in figure 4 with details on codes in Appendix B, figure 10. Figure 4 indicates the process of defining, implementing, and deploying consensus rules. In Bitcoin's case, these changes are usually motivated by some *issues* which prevent the implementation from providing the full service envisaged in Nakamoto's white paper or code. The discovery of an issue leads to *development* before moving on to *deployment*. On the outskirts of the process lays the possible influence of *human error* and *social/political* (governance). Human errors may disturb development or deployment, where mistakes and errors typically occur during development and violations occur in deployment.

The subsequent sections describe how the nine *features* for deployment were derived from the codes applied during analysis. These features are used in nine different combinations to perform deployment using different *deployment techniques* that answer RQ1. These deployments may be influenced in negative ways by governance and errors, presented in ten different lessons as described in the section on *lessons learned* to answer RQ2.

4.1 Results of RQ1 (deployment techniques)

4.1.1 Fork types

Common terminology [39] describing different consensus rule changes in a blockchain distinguishes between hard and soft forks. However, when considering the low-level details of a rule change, these concepts become too general and fail to provide an accurate description. For instance, a hard fork has been established as a term for rule changes that will result in a permanent chain split. However, this can also happen in a soft fork if a minority deploys the change. Besides soft and hard forks, Zamyatin et al. [33] also describe bilateral and velvet forks. The last category of *velvet* forks is irrelevant in this case because it does not require network-wide agreement, making deployment a non-event. Their terminology was adopted to accurately distinguish **fork types** and focus on the following specifications:

- *Expanding* - changes that make previously illegal actions legal (commonly referred as a hard fork).
- *Reducing* - changes that restricts the set of valid actions (commonly referred as a soft fork).
- *Bilateral* - changes which deem all previous legal actions as illegal as well as expanding the rule set (commonly referred as a hard fork).

4.1.2 Deployment codes and features

This section introduces each analytical code in italic before presenting the features in bold. The features for deployment are **Deployment strategy, Fork type, Chain split risk, Parallel, Standard, Signal, Inclusive, Threshold** and **Trigger**, the code-feature relation can be seen in figure 5.

4.1.2.1 Code: *Compatibility*: The **deployment strategy** feature defines whether nodes 1) depend on each other to coordinate the timing of an upgrade, i.e., miner-activated strategy. 2) The upgrade is forced regardless of miners' promised support, i.e., a user-activated strategy. Or 3) Deployment must be forced due to an imminent issue, i.e., emergency-activated strategy.

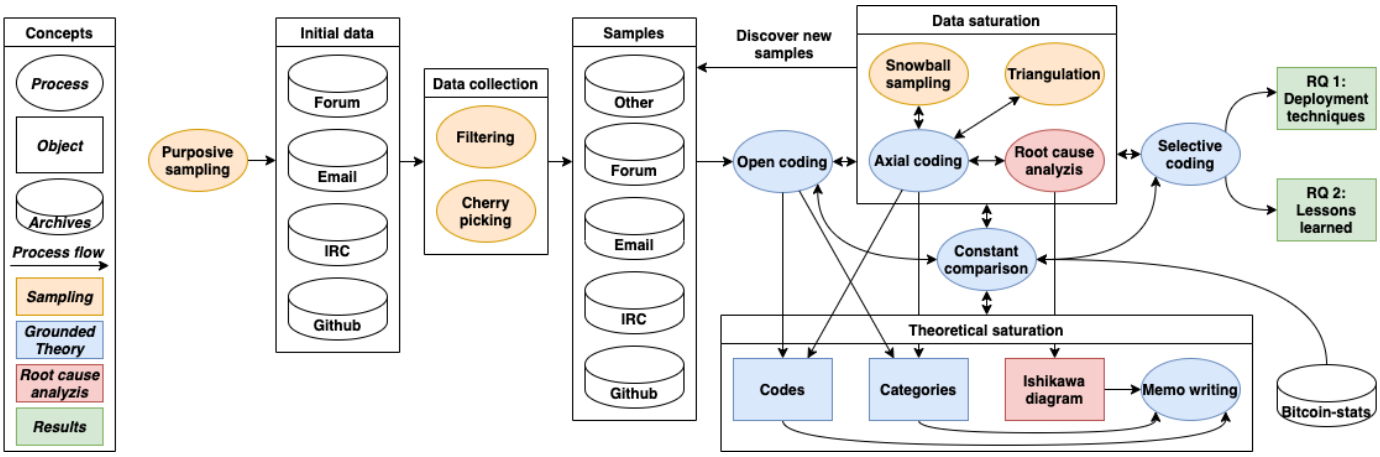

Fig. 3. Method process model

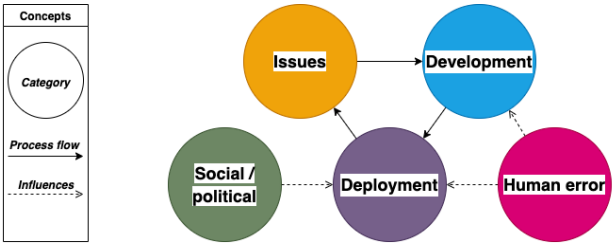

Fig. 4. Categories from the GT analysis and their relations.

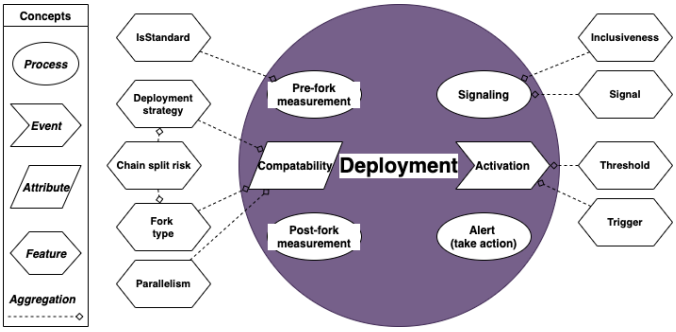

Fig. 5. Feature-code realations.

Whenever deploying rule changes to a blockchain, one must consider the compatibility between new and old versions by understanding the **fork type** of the implementation. The relevant fork types are expanding, reducing, and bilateral. The main difference is that a reducing fork will be backwards-compatible, allowing it to be enforced by the network with a super-majority of supporting hash power. Therefore, a reducing fork can be desirable as miners can keep the network consistent without relying on the whole network to perform the deployment. As listing 2 emphasizes, BTC developers usually look for ways to implement changes as reducing forks since they have desirable compatibility attributes and are easier to digest for the network and the community.

```
"I belief we shold flesh out luke-jr's idea for
cleanly deploying segregated witness in bitcoin as
a soft fork and see what that looks like."
```

Listing 2. Gmaxwell 2015-11-04 IRC: #bitcoin-dev

We propose a total of nine possible *deployment techiques* by combining the three fork types and three deployment strategies, as further described in section 4.1.3. These deployment techniques have an inherent **Chain split risk** feature, which indicates how likely a prolonged chain split is. Applying further deployment features can sustain chain splits' potential risk, length, and impact.

Another compatibility issue is whether it is possible to perform several deployments in **parallel**. Deployments can be conducted in parallel if the rule changes are isolated and the deployment attributes are independent. This feature is not by default and must be specified in the software implementation. Listing 3 shows Bitcoin adopting this feature after realizing that non-parallelism could become a problem.

```
"BIP 34 introduced a mechanism for doing
soft-forking (...). As it relies on comparing
version numbers as integers however, it only
supports one single change being rolled out at
once, requiring coordination between proposals, and
does not allow for permanent rejection: as long as
one soft fork is not fully rolled out, no future
one can be scheduled."
```

Listing 3. Pieter Wuille et al. 2015-10-04 Github, BIP9

4.1.2.2 Code: *Pre-fork measurement*: Pre-fork measurements can be used to harden the deployment process, mainly by preparing the blockchain to activate future reducing changes. The hardening can be performed by utilizing undefined elements such as opcodes or version numbers in blocks and transactions. For instance, a handful of undefined opcodes were deployed early into Bitcoin after initialization as an expanding fork (BTC 0.3.6) and allowed new opcode definitions to specify the undefined opcodes as reduction forks later. These were used for reducing changes such as BIP16 and BIP65. An example of utilizing an undefined opcode can be seen in listing 4.

```
"(...) OP_EVAL == OP_NOP1 can be safely rolled out
as soon as 50% of the miners upgraded"
```

Listing 4. Sipa 2011-10-02 IRC: #bitcoin-dev

In addition to the consensus rules, nodes can utilize relay policies to specify what transactions they will include in their blocks and whether they are relayed to other nodes. This can be seen as softly enforced rules that can be applied at any rate before activating new consensus rules, allowing individual miners to avoid unwanted or experimental be-

haviour. Blocks and transactions accepted under this policy are called standard. A deployment conducted by gradually changing the policies before or after the rule deployment will be recognized with the **standard** feature.

More aggressive enforcement of standard policies can be conducted by feather-forking, also known as block discouragement. Miners can use this approach to refuse to mine on top of blocks they do not appreciate. This gives an incentive to avoid these kinds of blocks as they have a higher chance of being orphaned together with the block reward.

```
"A feather-fork is when a miner refuses to mine on
any chain that includes a transaction it doesn't
like in the most recent several blocks."
```

Listing 5. Socrates1024 2013-10-17 Forum, ID: 312668

4.1.2.3 Code: *Signaling*: A **signal** is a feature used to signal the intention to upgrade and enforce new consensus rules in Bitcoin that is usually represented by the version bits in the block header or a string in the coinbase transaction message. Other implementations of on-chain signals observed are multi-signature commitments to the chain as implemented in Dash [40]. The signals make deployment more predictable and allow the measurement of total hash power support on the network. Listing 6 describes the first approach to signalling to deploy pay-to-script-hash.

```
"when 50% of the last N coinbases contain "I
support FOOFEATURE", it's enabled"
```

Listing 6. Luke-jr 2011-10-02 IRC: #bitcoin-dev

It is preferred to depend on miners to signal readiness on-chain since that provides confidence that a significant portion of miners will behave according to the new rules. However, a signal can also come in other forms, such as a verbal agreement. This happened once in Bitcoin's history during the deployment of BIP 30 (listing 7). The downside is that the actors who are unaware of this agreement might accept previously valid blocks and transactions without knowing they were breaking the new rules.

```
"<gavinandresen> luke-jr: you're a mining pool
operator, would you be willing to coordinate with
the other big pools to get this fixed quickly[?]"
```

Listing 7. gavinandresen 2012-02-17 IRC: #bitcoin-dev

The signals are most helpful on-chain, where they can be interpreted by validating nodes to coordinate an upgrade. They can also be used to exclude blocks mined by non-signalling nodes to persuade them to upgrade. This behaviour is defined by the **inclusive** feature. An inclusive fork will continue to append blocks from miners that do not intend to validate by the new rules. In contrast, an exclusive fork will stop accepting blocks from miners who do not show intention to validate by new rules. This restriction can be lifted after a certain time or if the deployment fails.

4.1.2.4 Code: *Alert*: Nodes intending to upgrade can issue alerts on the peer-to-peer network to warn other nodes of coming or ongoing upgrades. Increased awareness will increase the chance of consensus when deploying an upgrade. Alerts are not considered a deployment feature since they happen off-chain and are not part of the deployment mechanism. Listing 8 shows how alerts provide maintenance information to nodes in the network.

```
"Alerts will be sent to pre-0.8 releases over the
next two months, telling people to either upgrade
or create a DB_CONFIG file so they can handle large
blocks. After May 15'th, blocks up to 1MB large
will be allowed again."
```

Listing 8. gavinandresen 2013-03-16 Github, Pull: 2373

4.1.2.5 Code: *Activation*: Activation and enforcement of consensus rule changes can happen in stages, which can be controlled by the **threshold** feature. One deployment technique may implement several thresholds. For instance, the first threshold enforces rules for all signalling nodes. The second threshold enforces the rules for all nodes. Having several thresholds is a tradeoff. On the one hand, the first threshold incentivises miners to stay true to their intention of validating by the new rules. On the other hand, every threshold is a potential trigger for consensus failure. This is possibly the reason that Bitcoin ceased using two-stage activation using ISM (IsSuperMajority) [18]. The activation thresholds are most relevant for miner-activated strategies because they rely on coordination with other nodes. The thresholds should be at least the super-majority (>50% in Bitcoin), ensuring enforcement on the longest chain.

When the threshold is reached, the **trigger** feature will enforce activation. The actiavtion is triggered dynamically, staticly or instantly. It is desirable to evaluate using a rolling window (dynamic height) to decide the timing for miner-activated strategies. A rolling window trigger will determine the amount of support based on a number of recent blocks. The most primitive trigger can be based on a static flag day (FD) or block height (BH), as used for user-activated strategies and demonstrated by Nakamoto (listing 9). Instant triggers are utilized in the urgency of an emergency and are adopted as soon as they are rolled out to prevent or resolve exploits or consensus fauilures. The static and trigger provides no guarantee that a super-majority of miners will prevent a chain split when the rules activate.

```
"if (blocknumber > 115000)
   maxblocksize = largerlimit
It can start being in versions way ahead, so by the
time it reaches that block number and goes into
effect, the older versions that don't have it are
already obsolete."
```

Listing 9. Satoshi 2010-10-03 Forum, ID: 1347

4.1.2.6 Code: *Post-fork measurement*: Post-fork measurements expand upon emergency-activated strategies and relate to measures that can be taken to reduce the impact of a consensus failure. This stage aims to discard the conflicting chain and coordinate nodes to reorganize the chain and agree on a common history. The most naive approach to solving a chain split is to ask nodes to either downgrade or upgrade their nodes in the same manner, starting from the same block height. An issue with applying the naive approach with no further due is that it might result in a persistent chain split before the valid chain becomes the longest. Miners that fail to act will prolong the issue by expanding the conflicting chain (listing 10).

```
   "If you're unsure, please stop processing
   transactions."
```

Listing 10. 2013-03-12 Pieter Wuille Email: bitcoin-dev

4.1.3 Deployment techniques

Deployment techniques are meant to ensure consistency and predictability for the parties involved in the deployment. The deployment techniques are summarized into nine categories, shown in table 1. The techniques are defined as a combination of two deployment features: 1) The fork

type (*expanding*, *reducing*, or *bilateral*) and 2) the deployment strategy (*miner*, *user*, or *emergency*). For each deployment technique, the optimal combination is presented for the remaining features to reduce the risk and impact of a chain split. In blockchains with immediate finality [17], there is unlikely to be chain splits unless there is a rollback. In those cases, the features will reduce the risk and impact of nodes failing to participate in consensus.

The only feature with consistent behaviour across the deploy techniques is parallel because it can always be useful to allow several deployments in flight simultaneously. The features in table 1 are highlighted as **essential**, *useful*, or insignificant (-). An overview of Bitcoin's evolution over time and the deployment features utilized are shown in table 2. The following paragraphs present the nine deployment techniques and three special cases: Temporary reduction fork, hybrid deployment and non-deterministic fork.

4.1.3.1 Miner-activated reduction fork (MARF): The MARF deployment technique is unique in that it is the only technique with a low chain split risk when deployed using all the available deployment features. Like all miner-activated techniques, it should rely on coordination between miners in the network. The coordination is achieved by relying on signals from other miners to reach at least super-majority support to ensure that the network will reject new invalid blocks. However, a higher threshold is desirable to reduce the frequency and length of potential chain splits. A dynamic trigger ensures that the threshold is reached before triggering the activation.

Applying exclusiveness to MARF means that legacy blocks will be guaranteed to be orphaned at activation, increasing the risk of orphans and short chain splits. However, it also persuades more nodes to upgrade since they know they will be discarded, increasing the chance of reaching a high threshold. Further, one can utilize standardness to enable soft enforcement of the new rules long before activation. This greatly reduces the chance of upcoming invalid transactions being mined.

4.1.3.2 Miner-activated expansion fork (MAEF): Expansion forks will cause legacy nodes to deviate from patched nodes when the behaviour of new rules appears in blocks. Thus the chain split risk is high, and the network will only stay consistent with 100% adoption. Alternatively, a super-majority threshold can reduce the impact of a split by keeping patched nodes together on the new chain from the time of activation. With less than super-majority adoption, the patched nodes will continue to follow the legacy chain as new blocks violating the legacy rules will be discarded due to the longest chain rule. This can cause frequent splits of the chain depending on the adoption percentage, as shown in figure 6. The issue of low adoption can also be avoided by exclusiveness to enforce the discarding of all legacy blocks, causing patched nodes to follow their own path of the *valid* longest chain. The signal can be utilized to increase predictability, and standardness allows for an experimental phase after deployment where miners only utilize the expanded rules if they choose to take the risk. An example of a consensus rule in an experimental phase is the multi-signature opcode which was non-standard from BTC launch until BTC 0.6.0. Probably there was some doubt whether the multi-signatures were safe to use, or worked as intended, justified by the bug present in the opcode [41].

4.1.3.3 Miner-activated bilateral fork (MABF): Bilateral forks carry the property that patched nodes will never create valid blocks according to legacy nodes and vice versa. Therefore, it is only possible to avoid a chain split with 100% adoption. Choosing any activation threshold less than 100% carries less utility in limiting the impact of a chain split, and there can only be one split. However, it can be helpful to demand a certain amount of support to ensure that the patched nodes can provide sufficient security and reliable service for the patched network. Signals provide predictability, and the trigger should be dynamic to ensure that the timing corresponds to the desired threshold. The exclusion feature does not matter as it is an inherent property of bilateral forks. Additionally, standardness can be applied for soft enforcement to restrict certain functionality in an experimental phase after activation.

4.1.3.4 User-activated reduction fork (UARF): When moving over to the domain of user-activated forks, the deployment has a different objective. In contrast to keeping the network consistent, it is more important that the fork activate regardless. Therefore, such a fork does not require any threshold and should activate by a static trigger. Signals are useful to increase predictability. Furthermore, exclusion is essential since the updated consensus rules cannot be expected to reach a super-majority. Exclusion may cause a single chain split, while not excluding legacy nodes will cause chain splits every time the new rules are violated. The standard feature can be essential for UARF to discriminate against upcoming rule-breaking transactions. Feather-forking can persuade other nodes to upgrade by actively attempting to orphan legacy blocks.

4.1.3.5 User-activated expansion fork (UAEF): UAEF requires all nodes to upgrade to avoid a chain split and is mainly applied when expecting full network adoption with high confidence. This property was observed as the preferable deployment technique for consensus changes in both BCH and Ethereum. In these cases, the forks have usually held high or unanimous support from the community. Changes are implemented in different node distributions and are expected to be adopted by the time of activation. Signals can provide some utility by making the upgrade more predictable and allowing for exclusion. Exclusion must be applied if there is any doubt of super-majority adoption. Otherwise, the upgraded nodes might follow the legacy chain even after adoption, as it might be the longest valid chain, as illustrated in figure 6. However, if there is doubt, implementing a bilateral fork will be more beneficial in avoiding influence from legacy nodes, just like when BCH forked off BTC by UABF [21]. Standardness can allow for an experimental phase after deployment.

4.1.3.6 User-activated bilateral fork (UABF): The most outstanding example of a UABF is the activation of BCH. The fork became bilateral by demanding the first block produced after activation be larger than 1 MB. Hence, legacy nodes would never accept the patched chain, and patched nodes would never accept the legacy chain. The user-activated strategy does not evaluate any threshold for support to coordinate a certain network portion, and the trigger is static. Signalling is a valuable feature as it increases predictability. Exclusion is embedded in bilateral forks, and

TABLE 1
Deployment techniques. The notations indicate whether the features are required to reduce the chain split risk and the impact of a chain split: **Bold: Essential**, *Italic: Useful*, -: Insignificant. The abbrevations are: Miner-activated (MA), user-activated (UA), emergency-activated (EA), reduction fork (RF), expansion-fork (EF), bilateral fork (BF), and super-majority (SM)

Deployment strategy	Fork type	Chain split risk	Parallel	Standard	Signal	Inclusive	Threshold	Trigger	Example consenus rule change
MA	RF	Low	*Yes*	**Yes**	**Yes**	*Exclusive*	**SM**	**Dynamic**	Reduce max blocksize
MA	EF	High	*Yes*	*Yes*	**Yes**	*Exclusive*	**100%**	**Dynamic**	Increase max blocksize
MA	BF	High	*Yes*	*Yes*	**Yes**	-	**100%**	**Dynamic**	Increase max blocksize & set min blocksize >legacy blocksize
UA	RF	Medium	*Yes*	**Yes**	*Yes*	**Exclusive**	**None**	**Static**	Reduce max blocksize
UA	EF	High	*Yes*	*Yes*	*Yes*	**Exclusive**	**None**	**Static**	Increase max blocksize
UA	BF	High	*Yes*	*Yes*	*Yes*	-	**None**	**Static**	Increase max blocksize & set min blocksize >legacy blocksize
EA	RF	Medium	*Yes*	-	-	*Inclusive*	**None**	**Instant**	Reduce max blocksize
EA	EF	High	*Yes*	-	-	**Exclusive**	**None**	**Instant**	Increase max blocksize
EA	BF	High	*Yes*	-	-	-	**None**	**Instant**	Increase max blocksize & set min blocksize >legacy blocksize

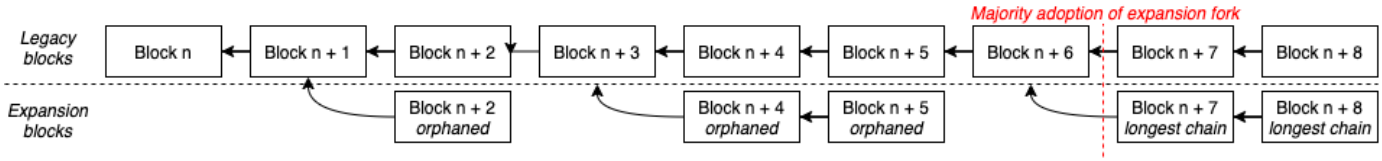

Fig. 6. Expansion forks can cause frequent chain splits before gaining super-majority adoption.

standardness allows for an experimental phase.

4.1.3.7 Emergency-activated reduction fork (EARF): Emergency-activated deployment strategies are required when the implementation does not work as specified, a consensus failure has already occurred, or might occur. One of the earliest cases, when the implementation did not work as intended, was seen in BTC 0.3.10 with the overflow bug where a seemingly valid transaction could be created to generate additional bitcoins. An example of a consensus failure was when BTC 0.8.0 deployed a new database, and it caused a chain split. The third case, a potential exploit, can be illustrated by the inflation bug in version 0.14.0, which was discovered before being exploited. All of these deployments were EARFs. It is useful for EARF deployments to be inclusive to allow unpatched nodes to reorganize and generate blocks on the valid chain originating from the reduction fork when it becomes the longest. EARF deployments in Bitcoin have been inclusive, which is the default behaviour.

An interesting observation in BTC 0.3.10 and 0.8.0 is that miners had performed a rollback of blocks, deviating from the longest valid chain rule and Bitcoin's immutability property to reach consensus. First, the overflow bug was so severe that the consensus rules had to be changed such that the chain containing the malicious transaction would be rejected. During the consensus failure caused by the database deployment, there was a need to downgrade nodes even though the new chain was the longest and valid according to the specification. That was because it was the most conservative approach to keep compatibility with old nodes and because many merchants and users were unlikely to follow the patched chain within a reasonable timeframe. Listing 11 shows that it was not obvious to downgrade and deviate from the longest chain rule. Furthermore, listing 12 highlights the reason to downgrade to the previous version.

```
"<Luke-Jr> gavinandresen: sipa: jgarzik: can we get
a consensus on recommendation for miners to
downgrade?
(...)
<gavinandresen> the 0.8 fork is longer, yes? So
majority hashpower is 0.8....
<Luke-Jr> gavinandresen: but 0.8 fork is not
compatible"
```

Listing 11. Luke-Jr & gavinandresen 2013-03-12 IRC: #bitcoin-dev

```
"Doesn't matter which chain is longer if a majority
of the people aren't on it. Breaking changes need
to be given lots of warning to be effective. Trying
to force everyone to use 0.8 would have only made
the situation worse. From the chat discussion, I
don't think mtgox was using 0.8. So trading at the
largest exchange would be halted until it could be
upgraded. If that doesn't sound disastrous, I'm not
sure what does."
```

Listing 12. nevafuse 2013-03-13 Forum, ID: 152470

There is no point in signalling in the urgency of emergency activation or waiting for a certain threshold. The triggering will happen naturally as soon as the bug in question triggers a consensus failure if the flaw is ever exploited. The exclusion will happen naturally as legacy nodes violating the rules of the EARF will be orphaned. In addition, standardness becomes unnecessary since the rule changes of an emergency fork require hard enforcement.

4.1.3.8 Emergency-activated expansion fork (EAEF): Deploying EAEF alone can be risky as it might not be widely adopted on a network basis, and miners might be reluctant to deploy a hasty and radical expansion of the consensus rules. Anything less than 100% adoption could cause a permanent chain split if never fully adopted. This encourages miners rather perform an emergency-activated reduction fork if feasible, as it is the safer alternative. Suppose a minority activates an EARF, and other nodes follow with different timings. In that case, they might split off from the legacy chain at different times, creating several chain splits until the patched chain becomes the longest, as

illustrated by figure 6. The figure shows how the patched nodes will keep jumping back to the legacy blockchain as long as that is the longest. As soon as the super-majority of miners work on the expanded blocks, that chain will become the longest one. However, legacy nodes will still work on the legacy chain as they do not see the expansion blocks as valid. It is possible that the BIP50 consensus failure caused by BTC 0.8.0 looked somewhat like the EAEF figure before making a persistent chain split, although that cannot be assessed without access to the orphaned blocks.

The continuous chain splits can be avoided by using the exclusion feature to invalidate the legacy chain. Another solution would be to make it a bilateral fork. However, an EAEF might be feasible in cases similar to the EAEF in version 0.3.7. A plausible reason for it to work without causing a chain split was that the expanded rules never occurred before a super-majority adopted the change.

4.1.3.9 Emergency-activated bilateral fork (EABF): This deployment technique has not been observed in any known upgrade. However, one could imagine the BCH fork being deployed with EABF as a reaction to revert the SegWit deployment. In that case, the patched chain would have to roll back to a block before the first SegWit-block was created and create a conflicting block. That block would be the instant trigger, while the remaining deployment features, except parallel, are insignificant.

4.1.3.10 Temporary reduction fork: There is a special case for the deployment of reduction forks (MARF, UARF, and EARF). In contrast to ordinary activation at time T, a temporary fork will activate at time T AND deactivate at time T+X. Temporarily reduction forks can be illustrated by an example from Bitcoin's legacy: The 1 MB limit was initially applied as a reduction fork. However, expanding that limit would not require an expansion fork if it was defined with an end-time, for instance, dating to 2022. Then the community would have years to find a solution or delay the issue by another temporary reduction fork before the end time. Legacy nodes can still validate all blocks created under the reduction fork, while patched nodes will know the start and end-time. BTC 0.8.1 demonstrated a temporary reduction fork shown in listing 13. The code defines a temporal reduction between 2013-03-21 and 2013-05-15 (line 2057 and 2058) where transaction IDs (TxIDs) are counted (lines 2062-2069) and the limitation is enforced (lines 2071 and 2072). The limit of 4,500 TxIDs was assumed to be low enough to avoid reaching the database lock limit of 10,000.

4.1.3.11 Hybrid deployment: Another special case in deployment is the deployment of different fork types together. This technique inherits the attributes of the most disruptive fork type in terms of chain split risk. That is in the following order: BF>EF>RF. Consider the case where all tree fork types are deployed together. The bilateral fork guarantees that the patched chain splits from the legacy chain patched nodes will never return to the legacy chain. Thus it is no longer meaningful to pay respect to whether there is super-majority adoption, inclusive fork or standardness because a bilateral fork neglects all these features. Hybrid deployment with combinations of expansion and reduction forks has become a relatively common practice in BCH, which performed hybrid deployments in BCHN 0.16.0, BCHN, 0.18.0, and BCHN 0.19.12. Combining several

```
2056 // Special short-term limits to avoid 10,000 BDB lock limit:
2057 if (GetBlockTime() >= 1363867200 && // start enforcing 21 March
         2013, noon GMT
2058     GetBlockTime() < 1368576000) // stop enforcing 15 May 2013
             00:00:00
2059 {
2060     // Rule is: #unique txids referenced <= 4,500
2061     // ... to prevent 10,000 BDB lock exhaustion on old clients
2062     set<uint256> setTxIn;
2063     for (size_t i = 0; i < vtx.size(); i++)
2064     {
2065         setTxIn.insert(vtx[i].GetHash());
2066         if (i == 0) continue; // skip coinbase txin
2067         BOOST_FOREACH(const CTxIn& txin, vtx[i].vin)
2068             setTxIn.insert(txin.prevout.hash);
2069     }
2070     size_t nTxids = setTxIn.size();
2071     if (nTxids > 4500)
2072         return error("CheckBlock() : 15 May maxlocks violation");
2073 }
```

Listing 13. Code snippet illustrating a temporary reduction fork from BTC 0.8.1 [42]

forks into one deployment is practical because it limits deployments where the network is more vulnerable.

4.1.3.12 Non-deterministic forks: The non-deterministic forks are best explained by the example of BIP50 and the upgrades deployed with BTC 0.8.0 and BTC 0.8.1. The implementations contained a MAX_BLOCK_SIZE of 1 MB. However, this rule was often overrun by the default database locks setting in pre-0.8.0 nodes that were too small to handle certain large blocks containing many transactions, although less than 1 MB. The problem would surface long before the consensus failure of BTC 0.8.0 because blocks used too many locks led to reorganisations. This caused a lot of nodes to run custom configurations. Listing 14 shows the problem surfacing and that some miners set custom lock limits.

```
"<TD> EXCEPTION: 11DbException
<TD> Db::put: Cannot allocate memory
<TD> bitcoin in ProcessMessage()
<TD> ProcessMessage(block, 5798 bytes) FAILED
<TD> received block 00000000000001c0a13e
<TD> REORGANIZE
(...)
<DrHaribo> sturles said he got out of memory errors
without being out of memory, and that adding locks
and lockers fixed it"
```

Listing 14. sipa, TD & DrHaribo 2012-03-10 IRC: #bitcoin-dev

Furthermore, the Berkeley Database would behave inconsistently depending on the underlying hardware. The result is that chain splits and stuck nodes might appear non-deterministically. Listing 15 describes how nodes running identical code might behave differently depending on how the blockchain is stored on disk.

```
"(...) contents of each node's blkindex.dat
database is not identical, and the number of locks
required depends on the exact arrangement of the
blkindex.dat on disk (locks are acquired per-page)."
```

Listing 15. Gavin Andresen 2013-03-20 BIP50

When the database was changed in BTC 0.8.0 the new implementation would handle the locks differently and always be able to handle the edge-case blocks. The legacy nodes with custom lock limits would also handle these blocks. On the contrary, a non-deterministic set of the vanilla legacy node implementations would regard these blocks as invalid, causing a chainsplit. The fork was non-deterministic because of the inconsistent behaviour among legacy nodes.

TABLE 2
Deployed rule changes in BTC and BCH. Star(*) = Forked repository. Triggers: BH = Block height, FD = Flagday, and DH = Dynamic height.

V	Consensus change	Deployment Strategy	Fork Type	Chain split risk	Parallel	Standard	Signal	Inclusive	Threshold	Trigger
BTC 0.1.6	Time based locking nLockTime	UA	RF	Medium	Yes	No	None	Inclusive	None	BH
BTC 0.3.5	CVE-2010-5137 & CVE-2010-5141	EA	RF	Medium	Yes	No	None	Inclusive	None	Instant
BTC 0.3.6	Disable/enable opcodes	EA	EF & RF *hybrid*	High	Yes	No	None	Inclusive	None	Instant
BTC 0.3.7	Separate scriptSig & scriptPubKey	EA	EF & RF *hybrid*	High	Yes	No	None	Inclusive	None	Instant
BTC 0.3.10	Output-value-overflow CVE-2010-5139	EA	RF	Medium	Yes	No	None	Inclusive	None	Instant
BTC 0.3.12	20 000-signature operation limit & 1 MB blocksize	UA	RF	Medium	Yes	No	None	Inclusive	None	BH
BTC 0.6.0	BIP30: Duplicate transactions CVE-2012-1909	MA	RF	Low	Yes	No	Offchain agreement	Inclusive	>50%	FD
BTC 0.6.0	BIP16: Pay-to-script-hash	MA	RF	Low	Yes	No	Coinbase vote	Inclusive	55%	FD
BTC 0.7.0	BIP34: Include block height in coinbase	MA	RF	Low	No	No	VersionBits	Exclusive	75%/95%	DH
BTC 0.8.0	BIP50: Migrate from Berkeley DB to LevelDB	UA	EF *non-deterministic*	High	Yes	No	None	Inclusive	None	Instant
BTC 0.7.0	BIP50: Rollback	EA	RF	Medium	Yes	No	None	Inclusive	None	Instant
BTC 0.8.1	BIP50: Database lock limit & Max TxID limit	UA	RF *temporarily*	Medium	Yes	No	None	Inclusive	None	Instant
BTC 0.8.1	BIP50: Relax database lock limit & Max TxID limit	UA	EF	High	Yes	No	None	Inclusive	None	FD
BTC 0.9.2	BIP42: 21 million supply	UA	RF	Medium	Yes	No	None	Inclusive	None	Instant
BTC 0.10.0	BIP66: Strict DER signature	MA	RF	Low	No	Yes	VersionBits	Exclusive	75%/95%	DH
BTC 0.11.2	BIP65: Check lock time verify	MA	RF	Low	No	No	VersionBits	Exclusive	75%/95%	DH
BTC 0.12.1	BIP68, BIP112, BIP113: Check sequence verify	MA	RF	Low	Yes	No	VersionBits	Inclusive	95%	DH
BTC 0.13.1	BIP141, BIP143, BIP146: Segregated Witness	MA	RF	Low	Yes	Yes	VersionBits service bits	Inclusive	95%	DH
BTC 0.14.0	CVE-2018-17144	UA	EF	High	Yes	No	None	Inclusive	None	Instant
BTC* 0.14.0	BIP148: Segregated Witness	UA	RF	Medium	Yes	Yes	VersionBits	Exclusive *temporarily*	None	FD
BTC* 0.14.1	SegWit2x/BIP91: Segregated Witness	MA	RF	Low	Yes	Yes	VersionBits	Inclusive	80%	DH
BCHN 0.14.5	Block size fork	UA	BF	High	Yes	No	None	Exclusive	None	FD
BCHN 0.16.0	LOW_S & NULLFAIL & difficulty adjustment	UA	EF & RF *hybrid*	High	Yes	Yes	None	Inclusive	None	FD
BCHN 0.17.0	Block size increase & enable opcodes	UA	EF	High	Yes	No	None	Inclusive	None	FD
BCHN 0.18.0	Various rule changes	UA	EF & RF *hybrid*	High	Yes	No	None	Inclusive	None	FD
BCHN 0.18.2	Fix CVE-2018-17144	EA	RF	Medium	Yes	No	None	Inclusive	None	Instant
BTC 0.16.3	Fix CVE-2018-17144	EA	RF	Medium	Yes	No	None	Inclusive	None	Instant
BCHN 0.19.0	Schnorr signatures & SegWit recovery	UA	EF	High	Yes	No	None	Inclusive	None	FD
BCHN 0.19.12	Schnorr signatures for multisig & script attributes	UA	EF & RF *hybrid*	High	Yes	No	None	Inclusive	None	FD
BCHN 0.21.0	SigChecks & Reversebytes opcode	UA	EF	High	Yes	No	None	Inclusive	None	FD
BCHN 22.0.0	ASERT Difficulty algorithm	UA	BF	High	Yes	No	None	Inclusive	None	FD
BCHN 23.0.0	Lift transaction chain limit & Multiple OP_RETURN outputs	UA	EF	High	Yes	No	None	Inclusive	None	FD
BTC 0.21.1	BIP341, BIP342, BIP343: Taproot	MA	RF	Low	Yes	Yes	VersionBits	Inclusive	90%	DH
BCHN 24.0.0	CHIP-2021-03 CHIP 2021-02	UA	EF	High	Yes	No	None	Inclusive	None	FD

4.2 Results of RQ2 (lessons learned)

All kinds of consensus rule changes in a blockchain can be a liability as they increase the attack surface. The deployment process itself can disrupt the community as conflicts arise. Lessons learned from Bitcoin deployments are synthesized to minimize the risk of future deployments in any blockchain. The GT analysis and root cause analysis derive these lessons as seen in the Ishikawa diagram in appendix B, figure 11. The issues and root causes are addressed under each lesson learned.

"Every time that you open up the door to changing the rules, you are opening yourselves up to attack"

Listing 16. 2020-08-03 Eric Lombrozo [43]

The human error categories [44] were used as codes in the GT analysis to classify the issues discovered in the root cause analysis. These are 1) Skill-based errors, i.e., execution failure: Slips and lapses. 2) Mistakes, i.e., planning failures: Rule-based (RB) mistakes and knowledge-based (KB) mistakes. 3) Violations: Routine, e.g., laziness and exceptional violations, e.g., sabotage. Table 3 shows the lessons derived, the impacted deployment features, their corresponding error categories and the affected Bitcoin versions.

4.2.1 L1: Missing transformation assurance

The most dangerous forks are those deployed by accident. They occur either because existing consensus rules are exploitable or new rules are deployed by accident. Accidental forks are not safely deployed using the deployment features and will have a high risk of a chain split. This error is seen as an RB mistake because developers misclassify the fork type feature of the given code change. The most obvious remedy is to perform extensive testing and review.

The lack of transformation assurance in Bitcoin has caused an accidental chain split on one occasion (BTC 0.8.0) and allowed a serious bug to enter the code (BTC 0.14.0). However, Bitcoin have never had an accidental chain split caused by compatibility issues cross node imlpementation, although the split in Ethereum's Berlin UAEF [45] demonstrates this. The fork type feature is relevant for this lesson because developers and node operators must understand whether the code changes they apply will alter consensus and what kind of fork type it is. Different techniques to provide assurances [46] of consensus rule transformation in blockchain are not widely adopted or explored.

"The review process is definetly a good idea, I dont know if it provides as much security as people assume it does. One thing that slip past one person may as well slip past ten people or whatever."

Listing 17. 2020-06-23 Luke-Jr [47]

Having several implementations can both cause and detect invalid transformation. Although BTC mainly relies on one implementation, there are many cryptocurrencies, such as BCH, using different implementations where all should follow the same consensus rules. Running testing on a test network with different implementations increases the chance of discovering transformation issues before deployment. However, having different implementations incease the risk of causing transformation issues (listing 18).

"Diversity is good and may help discover issues. But as Gavin was saying and as I like to point out: The most dangerous kind of failure in bitcoin isn't an implementation bug- any blockchain validation inconsistencies in widely deployed implementations are significantly worse than pretty much anything other than a full private key leak or remote root exploit... and are even harder to avoid."

Listing 18. Gmaxwell 2012-10-28 Forum, ID: 120836

4.2.2 L2: Improper reorganisation

Nodes must be prepared to handle reorganisation to coordinate everyone to work on the same chain immediately in case of an accidental split. Some nodes have been forced to redownload the whole blockchain, which the original slow initial block download (IBD) [48] made troublesome (BTC 0.3.10). Moreover, the database lock-limit caused stuck nodes during reorganisations prior to BTC 0.8.0 (see explanation in section 4.1.3.12). Some nodes would also wipe the existing mempool on reboot, making it harder to detect double spend attempts (BTC 0.8.0). Measurements should be taken to keep the current state of valid blocks and pending transactions when performing an emergency fork, enabling a swift recovery. The error leading to slow reorganisation could be a lapse in the case where node operators, in a weak moment, delete the whole blockchain on reboot and patch. It can also be a KB mistake where developers defining the code for reorganisations did not have the knowledge and experience to handle them properly. Listing 19 shows the issue of quickly reorganising the blockchain during the BTC 0.3.10 EARF.

knightmb, do you still have any of your monster network available to turn on to help build the new valid chain?

Listing 19. Insti 2010-08-15 Forum, ID: 823

4.2.3 L3: Improper human interference

Bitcoin's early history shows improper handling of deployment attributes. This happened in the pay-to-script-hash upgrade. The activation trigger feature is relevant here. The BTC developers set and moved the flag day trigger manually depending on whether the threshold was reached (BTC 0.6.0). The threshold was not met in time for the first flag day, and nodes had to update to change the new flag day. Some nodes did not catch this in time and lost track of the correct chain as an invalid pay-to-script-hash transaction was mined after the first flag day. The error can be seen as a RB mistake from the developers' side, who had faulty expectations for node operators. It could also be a KB mistake from the node operators' side if they were unaware of the changed flag day or a lapse in case they forgot to update in time. The error in the pay-to-script-hash deployment demonstrates that dynamic triggering must be incorporated into the software, not changed manually.

Deployment features should not be changed during deployment because each change acts as a fork by itself and is, therefore, a liability. In addition, developers should not alter ongoing deployment without giving time to review changes. That can be severe as it can allow the inclusion of flawed or ill-intended changes at the last minute.

4.2.4 L4: Too high thresholds

High thresholds are crucial to onboard hash power during deployment. The threshold feature is relevant in combination with the deployment strategy feature since miner-

TABLE 3
Lessons learned, impacted features, corresponding error categories and impacted Bitcoin versions.

ID	Lesson	Impacted features	Error categories	Negatvely affected versions
L1	Missing transformation assurance	Fork type	RB Mistake	BTC 0.8.0, BTC 0.14.0, BCHN 0.17.0
L2	Improper reorganisation	-	Lapse, KB Mistake	BTC 0.3.10, Before BTC 0.8.0, BTC 0.8.0
L3	Improper human interference	Trigger	Lapse, RB mistake, KB mistake	BTC 0.6.0
L4	Too high thresholds	Threshold, inclusive & deployment strategy	RB Mistake	BTC 0.13.1
L5	Deplyoing 'irreversible' changes	Fork type	KB Mistake	BTC 0.3.12
L6	Not prepared for forward compatibility	Fork type, standard & parallelism	KB Mistake	BTC 0.7.0, BTC 0.10.0, BTC 0.11.2
L7	Lacking knowledge regarding network dynamics	Signal	KB Mistake	-
L8	Insufficient damage control	-	KB Mistake	BTC 0.8.0
L9	Improper miner incentives to enforce new rules	-	Routine violation, Exceptional violation	BTC 0.10.0
L10	Insufficient incentices to review the code	-	Routine violation	BTC 0.8.0, BTC 0.14.0, BCHN 0.17.0

activated strategies utilize thresholds. The Segwit deployment (BTC 0.13.1) showed that high thresholds such as 95% is troublesome because it allows a >5% minority veto as shown in listing 20. The error can be seen as an RB mistake because SegWit was falsely considered non-controversial. The slow adoption of SegWit engaged the use of the less safe user-activated strategy to overthrow non-signalling nodes using the inclusive feature to exclude the opponents.

"Activation is dependent on near unanimous hashrate signalling which may be impractical and is also subject to veto by a small minority of non-signalling hashrate."

Listing 20. Shaolinfry 2017-04-06 Email: bitcoin-dev

BTC demonstrated some changes to avoid issues with high thresholds in the most recent Taproot upgrade (BTC 0.21.1). They changed the activation threshold to 90%, reduced the decision-making time, and intended to use user-activated deployment if the miner-activated deployment failed. Another method to cope with high thresholds is gradually decreasing the threshold towards the lower limit of super-majority. Dash's dynamic activation thresholds utilize this technique where the initial limit is 80% and is gradually reduced to 60% [49]. However, there is a tradeoff that lower thresholds are more likely to disrupt consensus.

4.2.5 L5: Deploying 'irreversible' changes

Changes conducted with reduction forks should be applied carefully as they might never be reverted if the nodes are reluctant to adopt future expansion forks. This makes the fork type feature relevant for this lesson. Nakamoto could probably never have imagined the fuzz caused by his 1 MB block size reduction that was created as a remedy for denial-of-service (BTC 0.3.12). So far, this reduction seems to be nearly irreversible in practice for the expansion-reluctant BTC community. Therefore it can be valuable to consider the fork type of temporary reduction forks when there is any doubt whether such a reduction fork should be permanent. The error is classified as a KB mistake because the developer did not foresee the future challenges of expanding the consensus rules. Listing 21 shows frustration for the 'irreversible' block size limit since the early days of Bitcoin.

"I'm very uncomfortable with this block size limit rule. This is a "protocol-rule" (not a "client-rule"), what makes it almost impossible to change once you have enough different softwares running the protocol. Take SMTP as an example... it's unchangeable."

Listing 21. Caveden 2010-11-20 Forum, ID: 1347

4.2.6 L6: Not prepared for forward-compatibility

Nakamoto implemented support for Bitcoin to be forward compatible. He created domains of undefined behaviour by initially defining block versions, transaction versions, and later OP_NOP opcodes (BTC 0.3.6). This is related to the fork type feature because it facilitates specifying future changes as reduction forks, which are safer. Most of the planned reduction forks in Bitcoin have depended on forward-compatibility. Forward-compatibility was further adopted when SegWit was created. The developers defined a 4-byte nVersion field to allow future changes to the script specification to be created as reduction forks (BTC 0.13.1, BTC 0.21.1). The error of not preparing for forward-compatibility can be seen as a KB mistake as developers might not be aware of the future compatibility issues. However, some people, the BCH community and many other blockchain projects (e.g., Ethereum and Dash) do not value compatibility between versions. They rather perform less safe expansion forks if that makes the end product more elegant.

"This is another problem that only exists because of the desire to soft fork. If "script 2.0" is a hard fork upgrade, you no longer need weird hacks like scripts-which-are-not-scripts."

Listing 22. Mike Hearn 2014-11-04 Email: bitcoin-dev

The standard feature facilitates forward-compatibility. This was done for all consensus rules dealing with malleability (BTC 0.10.0, BTC 0.13.1, BTC* 0.14.0, BTC* 0.14.1, BCHN 0.16.0 and BTC 0.21.1). All the rules changes related to malleability were already softly enforced by standardness. The standard nodes would minimize the success of malleability attacks by not including or relaying those transactions.

Additionaly, the parallel feature is relevant to forward-compatibility as it makes it possible to perform several deployments at the same time or sequentually. This was not the case for the first established deployment method ISM used in BTC 0.7.0, BTC 0.10.0 and BTC 0.11.2. These versions were deployed without the forward-compatibility and would not allow other deployments in parallel. Addi-

tionally, this technique permanently consumed versionBits such that they could never be used in a reduction fork again.

4.2.7 L7: Lacking knowledge regarding network dynamics

Some rule changes may require nodes to broadcast additional information to other nodes in the network. Like SegWit's extension blocks that legacy nodes would not relay. The worst-case outcome of this behaviour could be that the network would create partitions of nodes that could only validate blocks made within that partition. A potential error of network partitioning would be a KB mistake because of the lack of knowledge regarding network dynamics. As seen in listing 23 the peer-to-peer network relied on the signal feature by using a service bit for a node to signal the ability of providing the witness data (BTC 0.13.1).

```
"To ensure that the network is not partitioned and
that segwit blocks are being passed to segwit
enabled nodes, a Core 0.13.1 node will use its
outgoing connection slots to connect to as many
nodes with the NODE_WITNESS service bit as possible
(...)"
```

Listing 23. achow101 2016-11-26 Forum, ID: 1682183

4.2.8 L8: Insufficient damage control

Forks are necessary for the evolution of blockchains. As history has proven and Murphy's law will ensure, consensus failures will occur in the future. End-users and miners should take measurements to perform damage control. Past failures to perform these measurements would be a KB mistake because the actors did not know or did not have experience with chain splits. These measurements would be to detect a chain split and suspend transactions or increase the amounts of confirmations required. In the past, merchants have been subject to double-spend attacks, and pool funds have been drained by miners working on the chain that eventually orphaned (BTC 0.8.0, listing 24). In Bitcoin, there are mechanisms for detecting chains splits. Additionally, one can run nodes with different versions to monitor that they stay on the same chain. Listing 25 discusses some ways to perform damage control in case of a consensus failure.

```
"I've lost way too much money in the last 24 hours"
```

Listing 24. Eleuthria 2013-03-12 IRC: #bitcoin-dev

```
"So I think the only way Mallory gets free beer
from you with segwit soft-fork is if:
- you're running out of date software and you're
ignoring warnings to upgrade (block versions have
bumped)
- you've turned off standardness checks
- you're accepting low-confirmation transactions
- you're not using any double-spend detection
service"
```

Listing 25. Erisian 2015-12-18 Email: bitcoin-dev

Replay attacks can be performed by re-broadcasting transactions from one chain to another in the event of a permanent chain split. To prevent this attack, one of the chains should to implement replay protection [8]. BCH implements replay protection for all planned consensus changes [50].

Some damage control can be prepared up front, e.g., by incorporating a kill switch mechanism [51] to activate an emergency rollback. However, this kill switch mechanism have an increased risk of centralization and foreign interception if it is held by a single person or closed community.

4.2.9 L9: Improper miner incentives to enforce new rules

Even though miners give a signal for an upgrade in blocks, that does not guarantee that these miners will enforce the new rules. Simple-payment-verification (SPV) mining has become popular because less validation gives an advantage in the block race. The incentive mechanism in Bitcoin rewards the first valid block, and the tradeoff between the risk of not being first and the risk of being invalid may favor being first as it was seen in BTC 0.10.0 [4] (listing 26). The grace time between the time of reaching the threshold and the time of activation was added through BIP9 [19] was likely added because of this incident to give miners some time to ensure proper validation in time for activation.

```
"If there is a cost to verifying transactions in a
received block, then there is an incentive to *not
verify transactions*. However, this is balanced by
the a risk of mining atop an invalid block."
```

Listing 26. nathan 2015-07-11 Email: bitcoin-dev

This error is caused by routine, or even exceptional violations where miners generate blocks without performing validation. Some measurements exist to incentivise validation. For instance, Ethereum's future slashing mechanism [52] which discourage reckless behaviour. Alternatively, Dash incentivises validation by requiring collateral for master nodes [53], as well as giving them extra rewards.

4.2.10 L10: Insufficient incentices to review the code

Another incentive issue is in regards to reviewing code. All the actors in Bitcoin do benefit from having bug-free code deployed in the network to secure the value of the currency. However, testing and reviewing can be tricky, costly, and tedious. The average Bitcoin participant (e.g., end-users and miners) would not even have the skills to perform that task. The stakes might be high for anyone pushing code that affects the network badly, as it may harshly influence their reputation. At the same time, there is not enough incentive to spend substantial time and resources on secure development and code review. The lack of incentives could make developers lazy and errors are made by routine violations.

Many of the critical bugs contained in Bitcoin have been fixed since its conception, and new ones arise as developers make mistakes (BTC 0.8.0, BTC 0.14.0 and BCHN 0.17.0). However, these mistakes are not for developers to bear alone but for those who naively adopt flawed code. A project directly incentivising its development is Dash, where 10% of block rewards are allocated to development [54].

5 Discussion

5.1 Comparison with related work

The available literature on the evolution of Bitcoin mainly concerns socio-technical aspects of governance [55], [56] and their economics [57]. For instance, [55] addresses "how and when code development practices combine into a pattern of self-organizing." Deployment of consensus rules is slightly mentioned in the paper as a practice that can result in competing infrastructures. This research builds upon their work by showing how this can be technically carried out.

Kiffer et al. [8] evaluates the event of the Decentralised Autonomous Organization (DAO) hack in Ethereum and replay attacks. The case of Ethereum's split is relevant for

damage control. This paper provides a bigger picture by showing how one may end up with a network partition.

Abortable consensus [58], [59], [60], and this paper is similar in how they both look at how and when the network should switch to an arbitrary consensus algorithm. The main goal for abortable consensus is to gain performance when needed and increase fault tolerance when the network fails. This paper focuses on preventing and controlling failures during the deployment and transition.

The similarities between open source software and blockchain evolution are seen in the decision-making on a code repository level [61], [62]. Anyone is free to propose a change, and it is up to the maintainers of a repository whether they want to include that change. However, the decision is not only based on the maintainers' preference in blockchain but also on the opinion of the community, miners and possibly developers of other implementations of the same protocol. Even when a change is included in a code repository, it does not mean the network will adopt it.

The adoption rate of a new network protocol spans several years, does not require a specific threshold for adoption, and can tolerate different versions running in parallel [63]. In contrast, the requirements for running different versions in parallel are different since blockchain relies on abandoning old consensus rules by activating new ones. The similarity between BTC and TLS is that they value backwards compatibility to allow nodes running old protocols to be part of the network during and after an upgrade.

Comparing blockchain deployment techniques to techniques in distributed systems such as fast reboot, rolling upgrade, and big flip [32], the most significant difference is that blockchain is decentralised, and the network must reach a consensus before changes can be activated. This is done through unique deployment features such as standardness, signals, thresholds and inclusiveness.

5.2 Implications

To our knowledge, this research is the first to present a holistic overview of deployment techniques for runtime evolution in blockchains. The deployment processes of consensus rule changes in the blockchain are important as they can be both the cause and remedy of consensus failures. By generalizing the adoption logic of blockchain, this paper can contribute to the field of self-adaptive systems where processes for runtime evolution of new domain logic require further exploration [64].

Practitioners can utilize the contributions of this paper to perform consensus rule changes most predictably and safely. The lessons learned in Bitcoin are valuable to prevent history from repeating itself. These lessons can strengthen the security of blockchains, hinder direct financial loss and preserve a blockchain's value as a cryptocurrency.

5.3 Threats to validity

The sheer amount of data and the limited number of researchers dedicated to this project may raise questions about missing data or analysis. Regular cross-author discussions have evaluated the analysis and results to address this. All the samples used for the analysis are also available at [65]. By conducting rigorous data collection through snowballing and triangulation, the analysis has gathered a holistic picture of Bitcoin evolution and other blockchains outside of the initial domain. The results are also strengthened by combining GT with root cause analysis.

This study seeks to avoid bias by looking at other viable projects with similar attributes, such as BCH, Ethereum and Dash. Different actors have different concerns, and results are represented by diverse perspectives provided by those who have worked on Bitcoin over the last decade. The thoroughness applied in this study gives confidence that the results are applicable to different blockchain architectures.

6 Conclusion

Safe deployment of consensus rules in blockchains is vital to hinder failures causing financial losses for miners and end-users. The paper demonstrates an extensive study using the grounded theory approach, flexible coding and root cause analysis to address these issues. This study specifies nine deployment techniques for blockchain with nine different features. Additionally, the study shows how contention may arise during rule changes in Bitcoin, resulting in ten lessons learned. The findings bring novel insights to prosper safe evolution of blockchains.

7 Future work

The greatest challenge in blockchain evolution is regarding compatibility and transformation assurance [66]. Like adaptive systems, blockchain systems would benefit from identifying whether a change is a fork and what kind of fork it will be. Transformation assurance would significantly reduce the risk of deployment failures. Another aspect is to handle deployments in a multichain environment. Various research indicates that these environments will rely on middleware to relay actions cross chains. We believe that this middleware must be responsible for listening to and triggering changes based on the deployment techniques used in the attached chains. Further research might identify the implications of consensus evolution in such an environment.

Acknowledgments

This work is jointly supported by the National Key Research and Development Program of China (No.2019YFE0105500) and the Research Council of Norway (No.309494).

References

[1] A. V. Wirdum. (2020) The battle for p2sh: The untold story of the first bitcoin war. Bitcoin Magazine. [Online]. Available: https://bitcoinmagazine.com/technical/the-battle-for-p2sh-the-untold-story-of-the-first-bitcoin-war

[2] ——. (2017) The long road to segwit: How bitcoin's biggest protocol upgrade became reality. Bitcoin Magazine. [Online]. Available: https://bitcoinmagazine.com/technical/the-long-road-to-segwit-how-bitcoins-biggest-protocol-upgrade-became-reality

[3] G. Andresen. (2014) March 2013 chain fork post-mortem. Bitcoin Core. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0050.mediawiki

[4] Bitcoin.org. (2015) Some miners generating invalid blocks. Bitcoin.org. [Online]. Available: https://bitcoin.org/en/alert/2015-07-04-spv-mining

[5] B. Core. (2018) Cve-2018-17144 full disclosure. Bitcoin Core. [Online]. Available: https://bitcoincore.org/en/2018/09/20/notice/

[6] C. Harper. (2021) Open ethereum clients encounter 'consensus error' after berlin hard fork; coinbase pauses eth withdrawals. CoinDesk. [Online]. Available: https://www.coindesk.com/tech/2021/04/15/open-ethereum-clients-encounter-consensus-error-after-berlin-hard-fork-coinbase-pauses-eth-withdrawals/

[7] Bitcoin.org. (2013) A successful double spend us$10000 against okpay this morning. Simple Machines Forum. [Online]. Available: https://bitcointalk.org/index.php?topic=152348

[8] L. Kiffer, D. Levin, and A. Mislove, "Stick a fork in it: analysing the ethereum network partition," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. New York, NY, USA: ACM, 2017, pp. 94–100. [Online]. Available: https://doi.org/10.1145/3152434.3152449

[9] A. Rashid, S. A. A. Naqvi, R. Ramdhany, M. Edwards, R. Chitchyan, and M. A. Babar, "Discovering "unknown known" security requirements," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 866–876. [Online]. Available: https://doi.org/10.1145/2884781.2884785

[10] S. Nakamoto. (2008) Bitcoin: A peer-to-peer electronic cash system. Bitcoin.org. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[11] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Advances in Cryptology — CRYPTO' 92*, E. F. Brickell, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 139–147.

[12] M. Jakobsson and A. Juels, *Proofs of Work and Bread Pudding Protocols(Extended Abstract)*. Boston, MA: Springer US, 1999, pp. 258–272. [Online]. Available: https://doi.org/10.1007/978-0-387-35568-9_18

[13] J. R. Douceur, "The sybil attack," in *Peer-to-Peer Systems*, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 251–260.

[14] M. Rosenfeld, "analysis of hashrate-based double spending," *arXiv preprint arXiv:1402.2009*, vol. abs/1402.2009, 2014. [Online]. Available: https://doi.org/10.48550/arXiv.1402.2009

[15] S. Sayeed and H. Marco-Gisbert, "Assessing blockchain consensus and security mechanisms against the 51% attack," *Applied Sciences*, vol. 9, no. 9, p. 1788, 2019.

[16] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, p. 382–401, jul 1982. [Online]. Available: https://doi.org/10.1145/357172.357176

[17] H. Knudsen, J. S. Notland, P. H. Haro, T. B. Raeder, and J. Li, *Consensus in Blockchain Systems with Low Network Throughput: A Systematic Mapping Study*. New York, NY, USA: Association for Computing Machinery, 2021, p. 15–23. [Online]. Available: https://doi.org/10.1145/3475992.3475995

[18] P. Todd. (2014) Op_checklocktimeverify. Bitcoin Core. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki

[19] P. Wuille, P. Todd, G. Maxwell, and R. Russel. (2015) Version bits with timeout and delay. Bitcoin Core. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0009.mediawiki

[20] S. Fry. (2017) Mandatory activation of segwit deployment. Bitcoin Core. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0148.mediawiki

[21] bitcoincash.org. (2017) Uahf technical specification. bitcoincash.org. [Online]. Available: https://github.com/bitcoincashorg/bitcoincash.org/blob/master/spec/uahf-technical-spec.md

[22] T. Mens, "Introduction and roadmap: History and challenges of software evolution," in *Software evolution*. Berlin, Heidelberg: Springer, 2008, pp. 1–11.

[23] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *Acm Sigact News*, vol. 33, no. 2, pp. 51–59, 2002.

[24] A. Tapscott and D. Tapscott, "How blockchain is changing finance," *Harvard Business Review*, vol. 1, no. 9, pp. 2–5, 2017.

[25] P. Treleaven, R. G. Brown, and D. Yang, "Blockchain technology in finance," *Computer*, vol. 50, no. 9, pp. 14–17, 2017.

[26] S. Saberi, M. Kouhizadeh, J. Sarkis, and L. Shen, "Blockchain technology and its relationships to sustainable supply chain management," *International Journal of Production Research*, vol. 57, no. 7, pp. 2117–2135, 2019.

[27] K. Korpela, J. Hallikas, and T. Dahlberg, "Digital supply chain transformation toward blockchain integration," in *proceedings of the 50th Hawaii international conference on system sciences*. Manoa Hawaii, USA: ScholarSpace, 2017, pp. 4182–4191.

[28] M. Mettler, "Blockchain technology in healthcare: The revolution starts here," in *2016 IEEE 18th international conference on e-health networking, applications and services (Healthcom)*. Munich, Germany: IEEE, 2016, pp. 1–3.

[29] T. McGhin, K.-K. R. Choo, C. Z. Liu, and D. He, "Blockchain in healthcare applications: Research challenges and opportunities," *Journal of Network and Computer Applications*, vol. 135, pp. 62–75, 2019.

[30] J. Kramer and J. Magee, "The evolving philosophers problem: Dynamic change management," *IEEE Transactions on software engineering*, vol. 16, no. 11, pp. 1293–1306, 1990.

[31] S. Ajmani, B. Liskov, and L. Shrira, "Modular software upgrades for distributed systems," in *ECOOP 2006 – Object-Oriented Programming*, D. Thomas, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 452–476.

[32] E. A. Brewer, "Lessons from giant-scale services," *IEEE Internet computing*, vol. 5, no. 4, pp. 46–55, 2001.

[33] A. Zamyatin, N. Stifter, A. Judmayer, P. Schindler, E. Weippl, and W. J. Knottenbelt, "A wild velvet fork appears! inclusive blockchain protocol changes in practice," in *International Conference on Financial Cryptography and Data Security*. Berlin, Heidelberg: Springer, 2018, pp. 31–42.

[34] K.-J. Stol, P. Ralph, and B. Fitzgerald, "Grounded theory in software engineering research: A critical review and guidelines," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 120—-131. [Online]. Available: https://doi.org/10.1145/2884781.2884833

[35] B. J. Oates, "Researching information systems and computing," 2006.

[36] N. M. Deterding and M. C. Waters, "Flexible coding of in-depth interviews: A twenty-first-century approach," *Sociological methods & research*, vol. 50, no. 2, pp. 708–739, 2021.

[37] J. Corbin and A. Strauss, *Basics of qualitative research: Techniques and procedures for developing grounded theory*. London, England, UK: Sage publications, 2014.

[38] K. Ishikawa, *Guide to quality control*. Hong Kong: Asian Productivity Organization, 1982.

[39] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton, New Jersey, USA: Princeton University Press, 2016.

[40] Dash. (2018) Introducing long living masternode quorums. Dash Core. [Online]. Available: https://www.dash.org/blog/introducing-long-living-masternode-quorums/

[41] B. Wiki. (2021) Op checkmultisig. Bitcoin Wiki. [Online]. Available: https://en.bitcoin.it/wiki/OP_CHECKMULTISIG

[42] G. Andresen. (2013) Before 15 may, limit created block size to 500k. Bitcoin Core. [Online]. Available: https://github.com/bitcoin/bitcoin/blob/v0.8.1/src/main.cpp

[43] E. Lombrozo. (2020) Bitcoin magazine's independence day: Activating taproot with eric lombrozo and luke dashjr. Bitcoin Magazine. [Online]. Available: https://www.youtube.com/watch?v=yQZb0RDyFCQ

[44] J. Reason, *Human error*. New York, NY, USA: Cambridge university press, 1990.

[45] C. Harper. (2021) Open ethereum clients encounter 'consensus error' after berlin hard fork; coinbase pauses eth withdrawals. CoinDesk. [Online]. Available: https://www.coindesk.com/tech/2021/04/15/open-ethereum-clients-encounter-consensus-error-after-berlin-hard-fork-coinbase-pauses-eth-withdrawals/

[46] R. d. Lemos, D. Garlan, C. Ghezzi, H. Giese, J. Andersson, M. Litoiu, B. Schmerl, D. Weyns, L. Baresi, N. Bencomo *et al.*, "Software engineering for self-adaptive systems: Research challenges in the provision of assurances," in *Software Engineering for Self-Adaptive Systems III. Assurances*. Berlin, Heidelberg: Springer, 2017, pp. 3–30.

[47] Luke-jr. (2020) Segwit/psbt vulnerability (cve-2020-14199) with luke dashjr — bitdevsla. BitDevsLA. [Online]. Available: https://www.youtube.com/watch?v=CojixIMgg3c

[48] Bitcoin.org. (2022) Running a full node. Bitcoin.org. [Online]. Available: https://bitcoin.org/en/full-node\#initial-block-downloadibd

[49] PastaPastaPasta. (2020) Dash core 0.16.0.1 release announcement. Dash Core. [Online]. Available: https://github.com/dashpay/dash/releases/tag/v0.16.0.1

[50] B. C. Node. (2017) Buip-hf digest for replay protected signature verification across hard forks. Bitcoin-ABC. [Online]. Available:

https://gitlab.com/bitcoin-cash-node/bchn-sw/bitcoincash-upgrade-specifications/-/blob/master/spec/replay-protected-sighash.md

[51] D. Core. (2021) sporks. Dash Core. [Online]. Available: https://docs.dash.org/en/stable/introduction/features.html\#sporks

[52] Ethereum.org. (2022) Proof-of-stake and security. Ethereum. [Online]. Available: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/\#pos-and-security

[53] I. Dash Core Group. (2022) Understanding masternodes. Dash Core. [Online]. Available: https://docs.dash.org/en/stable/masternodes/understanding.html

[54] I. Dash Core Group, Inc. (2021) Understanding dash governance. Dash Core. [Online]. Available: https://docs.dash.org/en/stable/governance/understanding.html

[55] J. V. Andersen and C. I. Bogusz, "Patterns of self-organising in the bitcoin online community: Code forking as organising in digital infrastructure." in *ICIS*. Seoul, South Korea: AIS, 2017. [Online]. Available: https://pure.itu.dk/portal//files/83566552/ICIS_revision_2017.pdf

[56] P. De Filippi and B. Loveluck, "The invisible politics of bitcoin: governance crisis of a decentralised infrastructure," *Internet Policy Review*, vol. 5, no. 4, 2016. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2852691

[57] J. A. Kroll, I. C. Davey, and E. W. Felten, "The economics of bitcoin mining, or bitcoin in the presence of adversaries," in *Proceedings of WEIS*, no. 11, Washington, DC. 37th and O Streets, Rafik B. Hariri Building, Washington, DC 20057, United States: ISE, 2013. [Online]. Available: http://www.infosecon.net/workshop/downloads/2013/pdf/The_Economics_of_Bitcoin_Mining,_or_Bitcoin_in_the_Presence_of_Adversaries.pdf

[58] W. Chen and B. S. Center, "Abortable consensus and its application to probabilistic atomic broadcast," Technical Report MSR-TR-2006-135, Tech. Rep., 2007.

[59] P.-L. Aublin, R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić, "The next 700 bft protocols," *ACM Trans. Comput. Syst.*, vol. 32, no. 4, jan 2015. [Online]. Available: https://doi.org/10.1145/2658994

[60] J.-P. Bahsoun, R. Guerraoui, and A. Shoker, "Making bft protocols really adaptive," in *2015 IEEE International Parallel and Distributed Processing Symposium*. Hyderabad, India: IEEE, 2015, pp. 904–913.

[61] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. Van Deursen, "Communication in open source software development mailing lists," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. San Francisco, CA, USA: IEEE, 2013, pp. 277–286.

[62] P. N. Sharma, B. T. R. Savarimuthu, and N. Stanger, "Extracting rationale for open source software development decisions—a study of python email archives," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. Madrid, ES: IEEE, 2021, pp. 1008–1019.

[63] R. Holz, J. Hiller, J. Amann, A. Razaghpanah, T. Jost, N. Vallina-Rodriguez, and O. Hohlfeld, "Tracking the deployment of tls 1.3 on the web: A story of experimentation and centralization," *ACM SIGCOMM Computer Communication Review*, vol. 50, no. 3, pp. 3–15, 2020.

[64] R. de Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cukic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K. M. Göschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezzè, C. Prehofer, W. Schäfer, R. Schlichting, D. B. Smith, J. P. Sousa, L. Tahvildari, K. Wong, and J. Wuttke, *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–32. [Online]. Available: https://doi.org/10.1007/978-3-642-35813-5_1

[65] J. S. Notland. (2022) Exported samples. Norwegian University of Technology and Science. [Online]. Available: https://folk.ntnu.no/jakobsn/Runtime%20Evolution%20of%20Bitcoin's%20Consensus%20Rules/exported%20samples.zip

[66] R. d. Lemos, D. Garlan, C. Ghezzi, H. Giese, J. Andersson, M. Litoiu, B. Schmerl, D. Weyns, L. Baresi, N. Bencomo *et al.*, "Software engineering for self-adaptive systems: Research challenges in the provision of assurances," in *Software Engineering for Self-Adaptive Systems III. Assurances*. Berlin, Heidelberg: Springer, 2017, pp. 3–30.

[67] B. Wiki. (2022) Consensus versions. Bitcoin Wiki. [Online]. Available: https://en.bitcoin.it/wiki/Consensus_versions

[68] B. Core. (2021) Bitcoin core integration/staging tree. Bitcoin Core. [Online]. Available: https://github.com/bitcoin/bips

[69] L. Foundation. (2022) Bitcoin-dev email list. Linux Foundation. [Online]. Available: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/

[70] S. M. Forum. (2021) Bitcoin forum. Simple Machines Forum. [Online]. Available: https://bitcointalk.org/

[71] B. Core. (2021) Pull requests. Bitcoin Core. [Online]. Available: https://github.com/bitcoin/bitcoin/pulls

[72] ——. (2021) Issues. Bitcoin Core. [Online]. Available: https://github.com/bitcoin/bitcoin/issues

[73] B. wiki. (2020) Irc channels. Bitcoin Core. [Online]. Available: https://en.bitcoin.it/IRC_channels

[74] Ninjastic. (2021) Ninjastic.space. Ninjastic.space. [Online]. Available: https://ninjastic.space

[75] T. F. S. Foundation. (2020) Gnu grep. the Free Software Foundation. [Online]. Available: https://www.gnu.org/software/grep/

[76] Erisian. (2021) bitcoin-core-dev. Erisian Consulting. [Online]. Available: https://www.erisian.com.au/bitcoin-core-dev/

[77] ——. (2021) bitcoin-dev. Erisian Consulting. [Online]. Available: https://www.erisian.com.au/bitcoin-dev/

[78] BuildingBitcoin. (2019) bitcoin-dev. buildingbitcoin. [Online]. Available: https://buildingbitcoin.org/bitcoin-dev/

[79] Erisian. (2018) bitcoin-dev. Erisian Consulting. [Online]. Available: http://azure.erisian.com.au/~aj/tmp/irc/

[80] P. Ellenbogen. (2016) Bitcoin's history deserves to be better preserved. Princeton University. [Online]. Available: https://freedom-to-tinker.com/2016/09/15/bitcoins-history-deserves-to-be-better-preserved/

[81] Atlas.ti. (2022) All-in-one research software. ATLAS.ti Scientific Software Development GmbH. [Online]. Available: https://atlasti.com/

[82] NVIVO. (2022) Qualitative data analysis software. QSR International. [Online]. Available: https://www.qsrinternational.com/nvivo-qualitative-data-analysis-software/home

[83] MAXQDA. (2022) Organize. code. analyse. present. VERBI GmbH. [Online]. Available: https://www.maxqda.com/

[84] J. Penrod, D. B. Preston, R. E. Cain, and M. T. Starks, "A discussion of chain referral as a method of sampling hard-to-reach populations," *Journal of Transcultural nursing*, vol. 14, no. 2, pp. 100–107, 2003.

[85] Bitcoin.org. (2022) Bitcoin is an innovative payment network and a new kind of money. Bitcoin.org. [Online]. Available: https://bitcoin.org/

[86] B. Research. (2017) Empty block data by mining pool. BitMex. [Online]. Available: https://blog.bitmex.com/empty-block-data-by-mining-pool/

APPENDIX A
TIMELINE

The timeline in figure 7 summarizes 34 consensus changing events including 24 changes in Bitcoin Core and 10 changes in Bitcoin Cash (represented by Bitcoin Cash Node (BCHN)) marked as gray boxes. The date and order of these changes are based on either the flag day/block for the activation, the date where the changes activated based on signal thresholds, or otherwise when these versions were released. The red boxes indicates some issue where the deployment was performed by emergency, caused a chain split, or other issues. These issuse are further assessed in section 4.2 on lessons learned.

APPENDIX B
RESEARCH IMPLEMENTATION

B.1 Purposive sampling

The initial step in collecting data related to the consensus changes in Bitcoin was to identify the consensus changing events in Bitcoin. An exhaustive list of all the consensus changes throughout Bitcoin Core's history is listed in the Bitcoin Wiki [67] and in table 4, which also highlight eventual issues. Further inspection reveals an overview of the main events and information such as the time, block height, version number, and deployment techniques. Development channels were identified as the most fruitful sources and their data was extracted to initiate the filtering process. However, the sample boundary was fluid such that samples from other domains were considered whenever they appeared. These could be domains such as other Bitcoin channels, announcements, news articles, magazines, videos, or other cryptocurrencies. The additional sources primarily strengthened the theories rather than expanding them, which shows the relevance of the development channels initially selected by purposive sampling.

The channels that were selected as initial data were the Bitcoin improvement proposals [68], the bitcoin-dev emails [69], the Development & Technical Discussion topic in the Bitcoin forum [70], the code repository (pull requests [71] and issues [72]), and IRC channels (#bitcoin-dev and #bitcoin-core-dev [73]). The total sample count is 1700 as depicted in figures 8 and 9. One sample corresponds to one thread (email, forum and Github), one proposal or one day of IRC messages.

B.2 Data collection

Filtering was applied to the initial data after acquiring an overview of consensus change events and corresponding BIP specifications. The first source of data that was considered was the emails. With a relatively compact overview of all the threads that have been made, it was considered viable to traverse through the titles to find relevant samples. After email samples were collected, the next step was the forum. However, the forum source proved to be large and challenging to sort through. Therefore the search was conducted in two stages: First, all the threads leading up to and surrounding the dates of each consensus incident were checked for relevance, which has been referred to as cherry-picking samples. Second, the forum was filtered with the help of a search tool targeting the Bitcoin forum [74]. The filtering was commenced using the search strings shown in table 4. This filtering approach was also applied for the Github search. The relevant BIPs could be found as they directly correlate to consensus changes and deployment techniques.

When considering the IRC channels, the initial plan was to focus on the meetings conducted in Bitcoin's history and limit the number of samples collected in the initial phase. However, after getting familiar with Bitcoin meetings of the past, it became clear that there were only a few meetings (four in total) before the developers decided on weekly meetings in fall 2015. Therefore a search was conducted by applying a few search strings that could indicate conversations about the issues of consensus changes that have taken place. These strings were "fork", "chain split", "stuck" and "reorg". With the use of Grep [75] the files were traversed all together, showing each related sentence. The whole log from each sample was collected whenever indicating some value or relevance. Considering the size of the resulting data set of IRC samples, they indicate that the search strings were accurate and broad enough to catch relevant samples. For instance, it revealed samples explaining how the deployment techniques were initially implemented and further developed to avoid failure. Additionally, the logs around important dates were cherry-picked and inspected, even if they included the search strings or not. Finding relevant samples also became simpler from the fall of 2015 as the weekly meetings could be collected. A challenge with the IRC logs has been that the logs are somewhat dispersed between different archives [76] [77] [78] [79]. Together with the inherent inconsistencies [80] within the archives, it became clear that all the different archives had to be considered when searching. The distribution of samples from the different sources can be seen in figure 8, while the distribution across each event can be seen in figure 9. The sum of samples across the events are larger than the sum of samples across sources because some samples relate to multiple events.

B.3 Grounded theory

Although Strauss' approach [37] was promising to answer the research questions, it became apparent that the grounded theory approaches do not describe much on how modern data analysis tools should be utilized most effectively when performing analysis with a large number of samples. To cope with this, we refer to flexible coding from [36], which explains ways that large sets of data can be collected and coded through qualitative data analysis software (QDAS) such as Atlas.ti [81], NVIVO [82] or MaxQDA [83]. These techniques were useful for getting an overview of the whole data set, defining the codes, grouping the data into different categories, and constantly comparing and considering those.

B.3.1 Open coding

The guidelines on flexible coding describe a viable approach for the purpose [36] of analysing large datasets using QDAS. The indexing approach was applied as a specific form of open coding designed to provide an overview of the

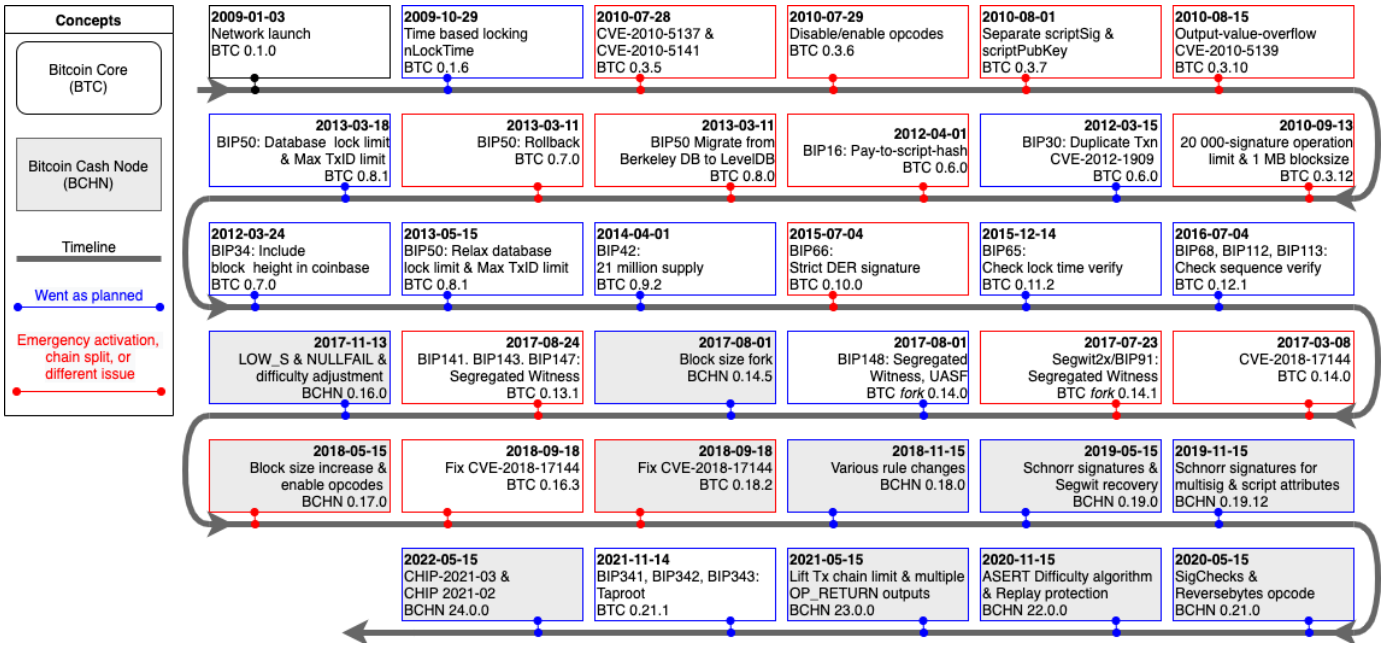

Fig. 7. Timeline of conensus changes in Bitcoin Core and Bitcoin Cash.

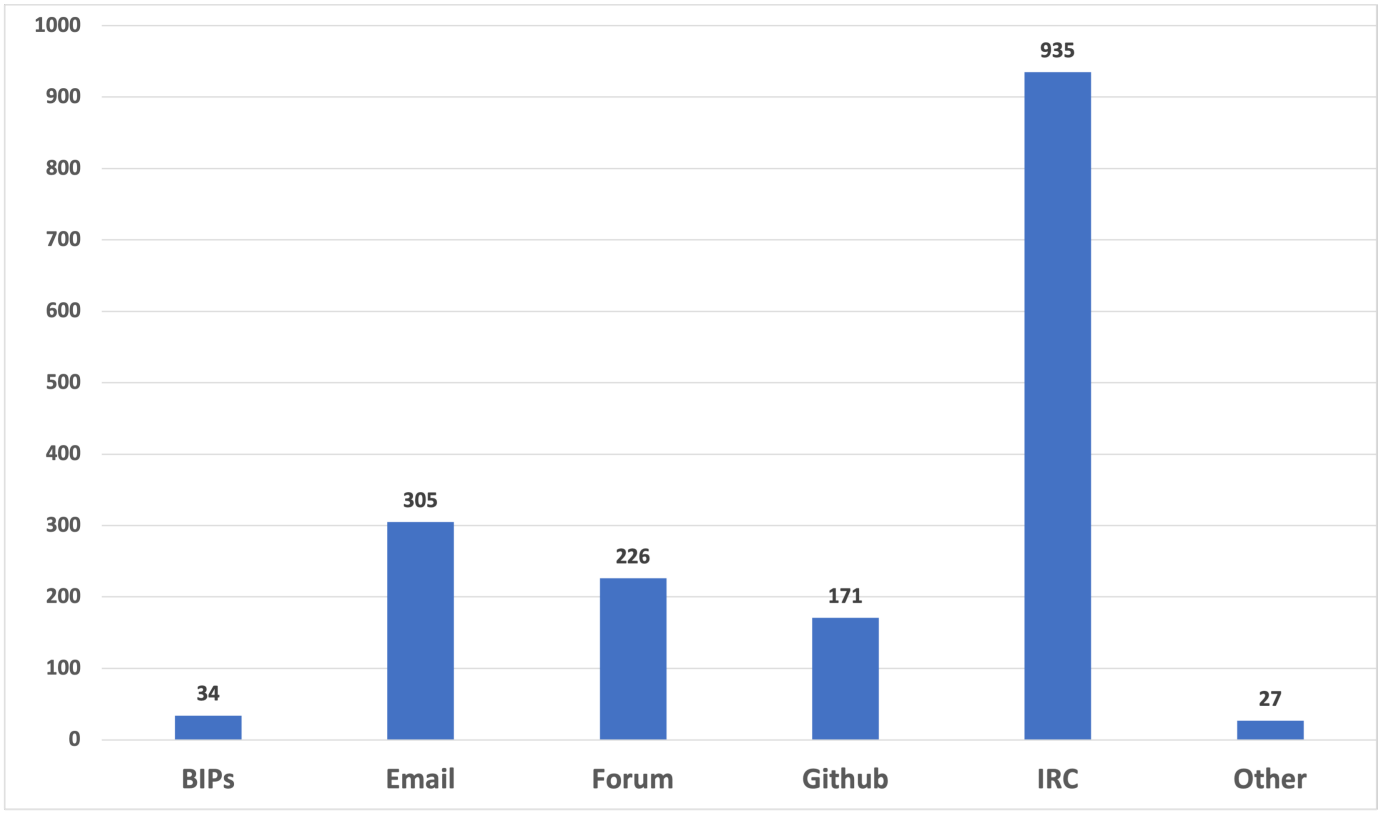

Fig. 8. Sample distribution among different sources.

TABLE 4
Overview of consensus changes in Bitcoin Core and applied search strings. **Bold** = blockchain specific issue.

V	Consensus change	Maintenance type	Issues	Search strings
BTC 0.1.6	Time based locking nLockTime	Corrective	No	nLockTime
BTC 0.3.5	CVE-2010-5137 & CVE-2010-5141	Corrective	No	CVE-2010-5137, CVE-2010-5141, 0.3.5
BTC 0.3.6	Disable/enable opcodes	Adaptive & preventive	Upgrading issues (bad dependency)	OP_CHECKSIG, OP_NOP, 0.3.6
BTC 0.3.7	Separate scriptSig & scriptPubKey	Corrective	No	scriptSig, scriptPubKey, 0.3.7
BTC 0.3.10	Output-value-overflow CVE-2010-5139	Corrective	**Slow adoption & long split**	overflow bug, 184, CVE-2010-5139, 0.3.10, 74638, 74 638
BTC 0.3.12	20 000-signature operation limit & 1MB blocksize	Corrective	**"Irreversible" change, source of controversy**	0.3.12, MAX_BLOCK_SIZE, MAX_BLOCK_SIGOPS, 79400 & 79 400
BTC 0.6.0	BIP30: Duplicate transactions CVE-2012-1909	Corrective	No	bip30, bip 30, duplicate transactions
BTC 0.6.0	BIP16: Pay-to-script-hash	Perfective	**Stuck Nodes, slow adoption, conflicting proposals,** new bug	bip12, bip 12, bip16, bip 16, bip17, bip 17, bip18, bip 18, OP_EVAL, P2SH, pay-to-script-hash, 0.6, 0.6.0
BTC 0.7.0	BIP34: Include block height in coinbase	Corrective	No	bip34, bip 34, 0.7, 0.7.0, 227835, 227 835
BTC 0.8.0	BIP50: Migrate from Berkeley DB to LevelDB	Corrective & perfective	**Accidental fork, long split, financial loss, double spend**	bip50, bip 50, leveldb, berkeley db, 225430, 225 430, max 4 500, reduce blocksize, lock limit & 0.8
BTC 0.8.1	BIP50: Database lock limit & Max TxID limit	Corrective	No	———
BTC 0.8.1	BIP50: Relax database lock limit & max TxID limit	Corrective	No	———
BTC 0.9.2	BIP42: 21 million supply	Preventive	No	bip42, bip 42, 21 million, 21 000 000, 13,440,000, 13440000, 13 440 000 & 0.9.0
BTC 0.10.0	BIP66: Strict DER signature	Corrective	**Short splits, false signaling**	bip66, bip 66, strict der signatures, 0.10.0, 0.9.5
BTC 0.11.0	BIP65: Check lock time verify	Perfective	No	bip65, bip 65, check lock-time verify, CHECKLOCKTIMEVERIFY, CLTV , 0.10.4 & 0.11.2
BTC 0.12.1	BIP68, BIP112, BIP113: Check sequence verify	Perfective	No	bip68, bip 68, bip112, bip 112, bip113, bip 113, Check sequence verify, Relative lock-time, CHECKSEQUENCEVERIFY, CSV, 0.11.3, 0.12.1
BTC 0.13.1	BIP141. BIP143. BIP147: Segregated Witness	Perfective, corrective, adaptive & preventive	**High risk for chain split, slow adoption, conflicting proposals, potential network parition &** complicated change	bip141, bip 141, bip143, bip 143, bip147, bip 147, segwit, 0.13, bip91, bip 91, bip148, bip 148, segregated witness, 481824 & 481 824
BTC 0.14.0	CVE-2018-17144	Perfective	new bug	-
BTC 0.16.3	Fix CVE-2018-17144	Preventive	No	-
BTC 0.21.1	BIP341, BIP342, BIP343: Taproot	Perfective & adaptive	No	-

initial data and effectively define and evaluate codes and categories. In practice, the data was initially indexed on a sample basis to highlight the essence of each sample. The process of indexing was constantly evaluated as new codes and categories emerged. Memos were created to understand the correlation between codes, categories, and events. One specific code was used to highlight notable quotes that had considerable impact on the results [36]. This code always overlap with some other code and is labeled "aha" to signal the aha-experience that these quotes represent. Quotes labeled with this code were consistently revisited and were candidates to present and support the content of this paper.

B.3.2 Axial coding

The axial coding phase was conducted by revising, combining, and splitting codes and categories. Some of them were combined with decreasing levels of granularity when the low-level details became unhelpful in addressing the research questions and increasing theoretical saturation. On the other hand, some codes were split to increase granularity and enhance insights. This stage also developed patterns and relations within the codes and categories.

The processes of indexing samples and identifying root causes revealed a collection of blockchain-specific issues as highlighted in table 4. A few issues in table 4 are related to well-known general software engineering practices, such as introducing bugs caused by insufficient reviewing and testing. Therefore, to further inspect the nature of blockchain-specific issues and to derive lessons learned, it became evident that the blockchain-specific issues such as chain split, network partitioning, adoption, controversal changes, stuck nodes, conflicting proposals and malicious behaviour should be analysed in depth.

B.3.3 Selective coding

At this stage, the analysis revealed the main categories of the data, their correlation, and the root causes of blockchain-specific issues. Therefore the process of selective coding

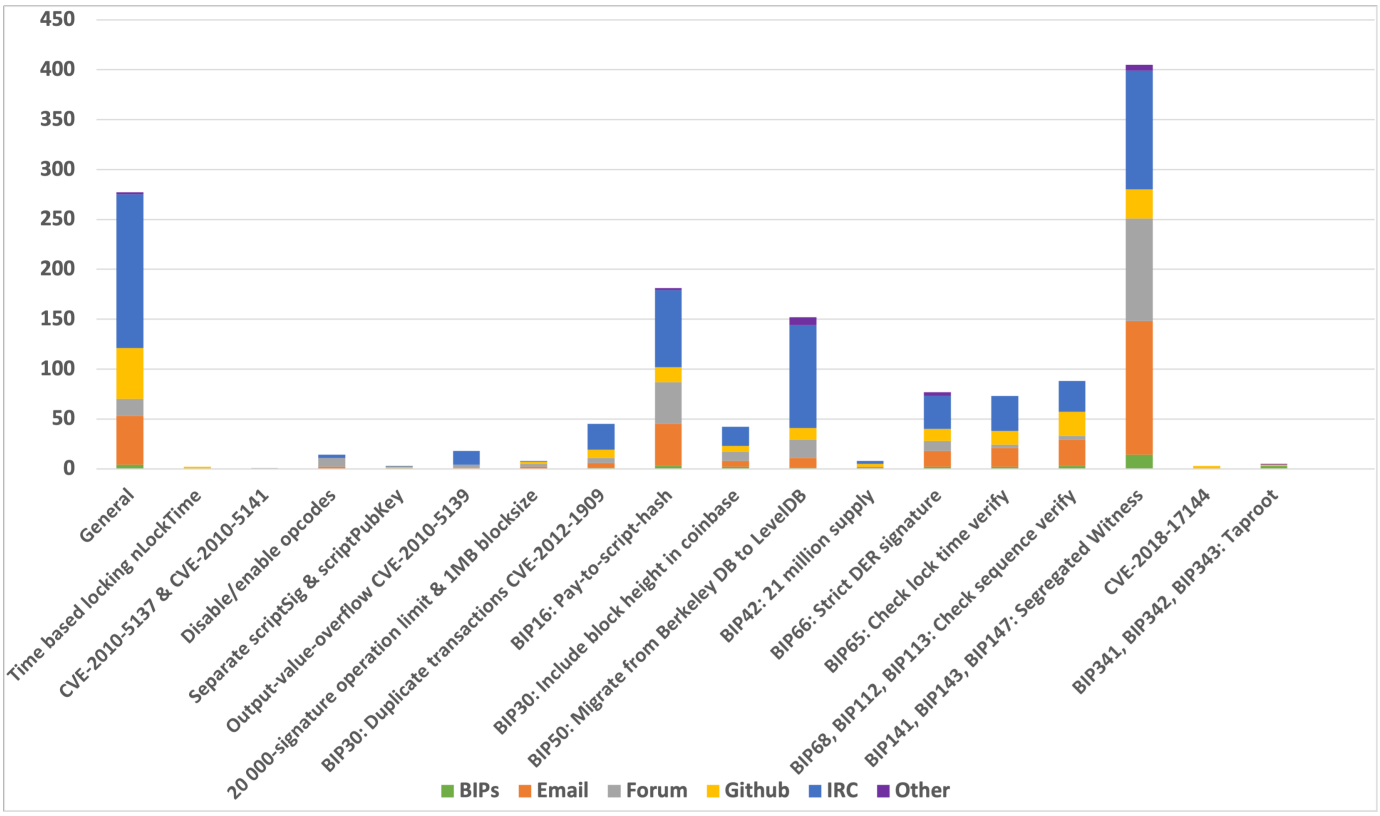

Fig. 9. Sample distribution among different consensus changes.

would focus on saturating these concepts. We realize the causalities of where deployment issues are rooted and where they surface. The lessons learned were revised to reflect this and the relevant deployment techniques to address all these issues are summarized. We further elaborated on Bitcoin and other blockchains' technocratic governance structures to reflect on the governance structures and their relation to the deployment techniques.

B.3.4 Constant comparison and theoretical saturation

The codes, categories and emerging theories were constantly compared by controlling whether they made sense in terms of the research questions and the objective domain. Samples that no longer gave more insights were stored in a group labeled "not interesting" to ensure that the most relevant and interesting logs were saturated first. It was assumed that the samples collected in the initial search could lead to an adequate theory in this paper. However, there was always a possibility that the current data set was too narrow. Therefore the techniques of snowballing [84] and data triangulation [37] were systematically applied.

The data saturation process led to additional findings not identified in the initial data collection phase. The findings could, for instance, be missing links from one of the resources used in the data collection, or it could be an article from the Bitcoin Project's website [85], a blog post, or a video. These new samples were systematically collected by applying the snowballing technique. Then they were also included in the data set if they were relevant to the research questions, which gave a significant amount of additional samples from resources other than the initially collected ones.

Data triangulation was applied between the different resources to see the same phenomena from different perspectives. It was possible to figure out that when something important happened in one place, there would probably be more to read about it from other samples. This gave a rich data set with multiple perspectives and helped when, for instance, the IRC chat had inconsistencies on the 2015 event (version 0.10.0) where 2015-06-03, 2015-06-04, and 2015-06-05 are missing. Having many other samples revealed how this specific consensus failure was caused by custom and lazy validation as well as spy-mining [86].

B.4 Root cause analysis

The appliance of Ishikawa diagrams [38] as seen in figure 11 further enhanced the analysis for RQ2 (lessons learned). The identified root causes were classified within the categories of human errors [44] to gain further insight. The different errors were applied as codes during grounded theory analysis as seen in section 4.2. The complementation of the grounded theory approach with root cause analysis gave higher confidence in covering relevant issues and gain an in depth understanding.

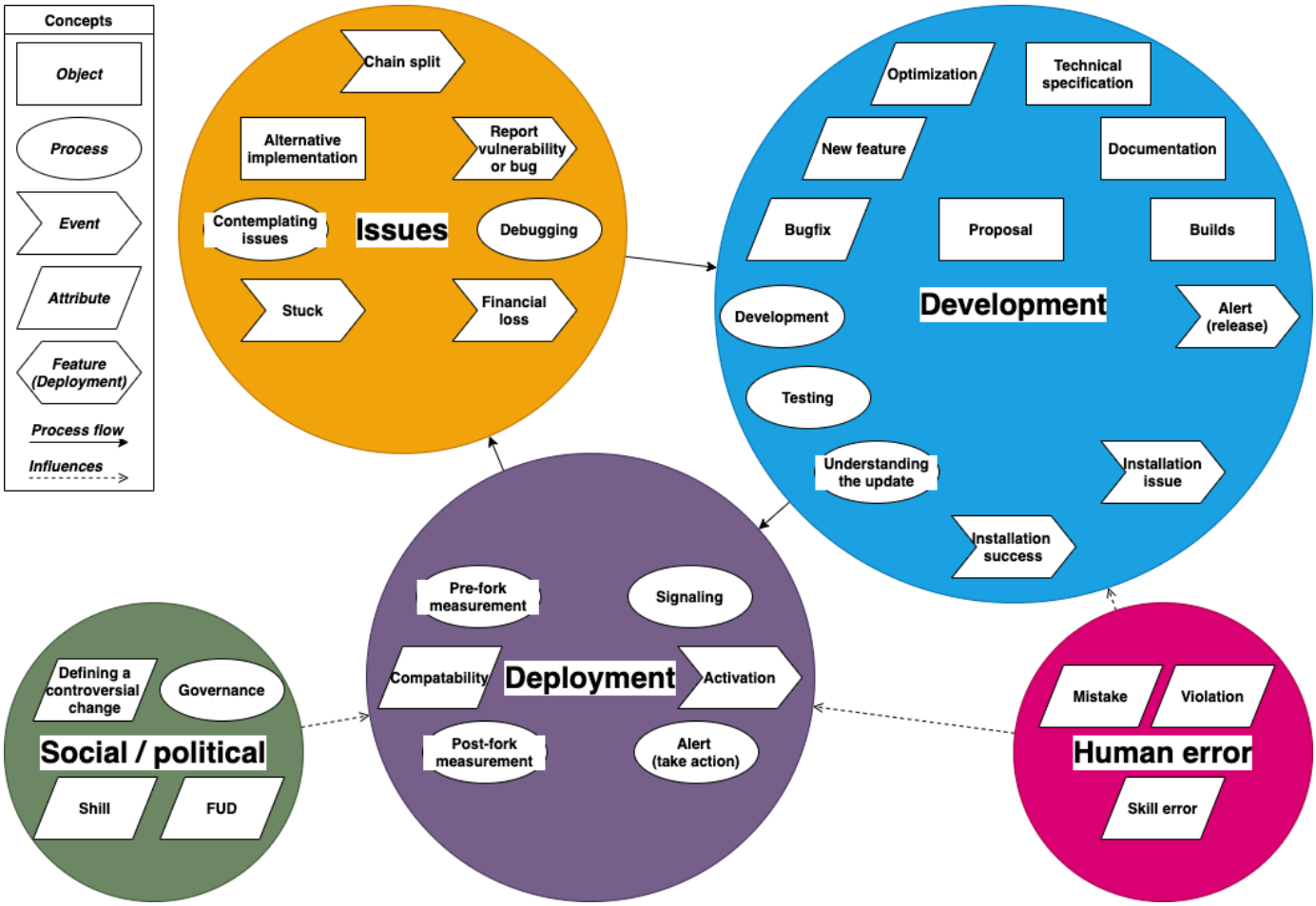

Fig. 10. Codes and categories

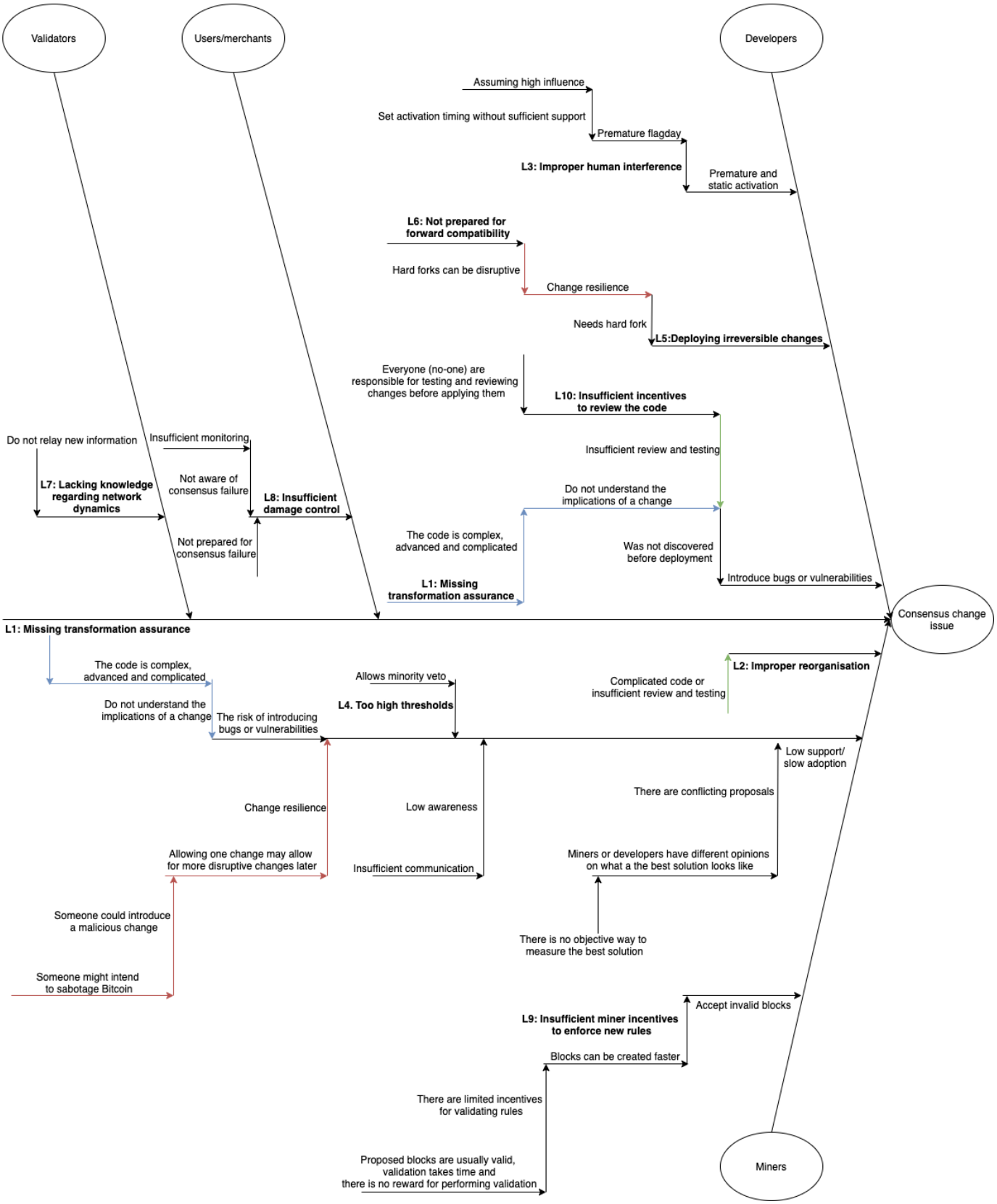

Fig. 11. Ishikawa diagram - root causes