Project Report: Formally Verified E-Voting INF-BSc-P15, Summer 2025

Robert Büttner, Patrick Janoschek, Ivana Kostadinovic, and Henrik Oback

Bachelor Programme in Computer Science
Faculty of Informatics and Data Science, University of Regensburg

Abstract. Ensuring fairness and correctness is the fundamental challenge for electoral systems, as both electronic and manual systems are susceptible to errors. This project aimed to develop a web-based e-voting application providing a mathematical guarantee of correctness for the implemented voting algorithms through formal verification. The system's core consists of two voting algorithms formally verified using the Dafny programming language: Borda Count with a modified Baldwin's tie-breaking method and Single Transferable Vote (STV) using Gregory's method for vote transfer. This verified core is embedded in a secure, server-side Blazor web application that supports public and private elections and ensures voter anonymity. This project shows that it is not only possible to theoretically prove that voting algorithms satisfy their formal specification, but also to bridge the gap between theory and the real world by making the algorithm accessible and usable for users of the web application.

1 Introduction

Elections are a cornerstone of modern societies. They provide a mechanism through which collective decisions can be made fairly and each individual can have a voice. Imagining a world without elections quickly reveals their indispensability. However, history also shows that the integrity of electoral systems is not always guaranteed. A voting process only fulfills its purpose if citizens can trust that it is conducted correctly, free from manipulation, fraud, or error.

Traditional elections rely on human actors and organizational structures. While these may operate with the best of intentions, mistakes - intentional or accidental - can never be fully ruled out. Electronic voting systems, in turn, reduce certain risks but introduce others. Although the execution of a program is deterministic, the software itself is written by humans and is therefore prone to design flaws or implementation errors. This raises a critical question. How can one guarantee the correctness and fairness of an election when both human and technical processes are susceptible to failure?

Our project addresses this challenge by developing a formally verified e-voting system. Using the programming language Dafny, we ensure that the implementation of the voting algorithms provably satisfies its formal specifications, thereby eliminating entire classes of potential programming errors. Moreover, since our formally verified source code is publicly available, voters and experts alike can inspect it, increasing transparency and trust in the system.

Formal verification is inherently complex and resource-intensive, particularly for larger systems, which explains why no complete formally verified e-voting system had existed prior to this project. Through considerable effort, our team succeeded in implementing and verifying two distinct voting algorithms: Borda Count and Single Transferable Vote (STV). On our accompanying website, users can conduct elections using either method while also accessing the Dafny-verified source code that ensures correctness.

1.1 Project Goals

The primary goal of this project was to design and implement a web-based electronic voting system that relies on formally verified algorithms written in Dafny. More specifically, our goals can be divided into the following sections:

1. **Implementation of verified voting algorithms:**
   - Provide two different voting algorithms, both formally verified in Dafny:
     - Borda Count with a modified Baldwin's method as a tie-breaking procedure.
     - Single Transferable Vote (STV) with Gregory's method for vote transfer.
2. **Accessible and secure web application:**
   - Develop a website through which users can set up an election by specifying parameters like candidates and the underlying voting algorithm.
   - Support two types of elections:
     - Public elections, which can be accessed and joined by any user.
     - Private elections, where participation is restricted to users directly invited via e-mail by the organizer.
   - Ensure that all votes are cast in a way that guarantees anonymity, so that individual choices cannot be linked back to specific voters.

3. **Transparency and verifiability:**

   - Make the Dafny source code publicly available on the website.
   - Ensure that users can either view the code directly online or download it for independent inspection.

4. **Teamwork and project management:**

   - Organize the collaboration so that responsibilities were shared fairly and communication remained efficient.
   - Practice collective decision-making and maintain weekly progress evaluations.

5. **Research and learning outcomes:**

   - Deepen our understanding of formal verification and its application to real-world problems.
   - Gain practical experience with the Dafny programming language and proof based development.
   - Explore the advantages and limitations of applying formal methods in complex domains such as voting systems.

6. **Documentation and dissemination:**

   - Produce clear documentation of both the implementation and the verification process.
   - Present the project and its outcomes to different audiences, including peers and instructors.

2 Background

2.1 Electronic Voting Systems

Electronic voting (e-voting) refers to the use of digital technologies for casting and counting votes in an election. While such systems promise efficiency and accessibility, they also raise significant concerns regarding security, transparency, and trustworthiness [1]. Unlike traditional paper-based systems, where the process is physically observable, e- voting requires voters to place trust in technical components such as software and hardware. This makes the correctness of the implementation and the verifiability of the process critical issues in both academic and political debates.

2.2 Formal Verification

Formal verification is the process of proving, using mathematical methods, that a program satisfies a given specification [6]. Unlike testing, which can only cover a finite set of input cases, formal verification provides guarantees about all possible program executions. This is particularly relevant in safety- and security-critical applications, such as aerospace, cryptography, and voting systems, where undetected errors can have severe consequences.

2.3 Dafny

Dafny is an imperative, object-based programming language with built-in support for specification and verification [12]. Programs written in Dafny can be annotated with preconditions, postconditions, and invariants, which the Dafny verifier checks automatically. By doing so, Dafny allows developers to construct programs that are not only executable but also mathematically proven to be correct. This makes it particularly well suited for applications like voting systems, where correctness and trust are of paramount importance.

2.4 Borda Count

The Borda Count is a positional voting method introduced by Jean-Charles de Borda in 1770 [4]. In this method, voters rank candidates in order of preference, and points are assigned based on position in the ranking. The candidate with the highest total score wins. Although simple and widely studied, the Borda Count is vulnerable to strategic manipulation and may fail to satisfy certain fairness criteria, such as the Condorcet criterion.

2.5 Baldwin's Method (Tie-breaking)

Baldwin's method is a procedure originally designed as an elimination method based on Borda scores [8]. In the context of this project, a modified version is used to resolve ties in the Borda Count. The principle is to iteratively eliminate the candidate with the lowest Borda score until a single winner remains, ensuring that ties are resolved consistently and deterministically.

2.6 Single Transferable Vote (STV)

The Single Transferable Vote (STV) is a preferential voting system designed to achieve proportional representation [9]. Voters rank candidates in order of preference, and votes are transferred according to these preferences as candidates are either elected or eliminated. This transfer mechanism allows minority preferences to be represented more fairly than in simple majority systems. STV has been adopted in several countries, such as Ireland and Malta, for parliamentary elections.

2.7 Gregory's Method (Vote Transfer)

Gregory's method is a specific algorithm for transferring surplus votes in STV elections [16]. Instead of transferring only a subset of ballots, as in other variants, Gregory's method redistributes each vote at a reduced fractional value. This ensures that all voters contribute equally to the outcome while preserving proportionality.

2.8 Relational Databases (SQL/MariaDB)

A relational database stores information in tables made up of rows and columns, where each row represents a record and each column represents a type of data. For example, one table might store voters, another candidates, and a third the votes they cast. SQL (Structured Query Language) is used to insert, retrieve, and manipulate this data in a structured way. Constraints, such as requiring unique voter emails or valid candidate IDs, help prevent errors and maintain consistent election data. MySQL was used during development and MariaDB will be used for deployment. Both are popular, reliable relational databases.

2.9 Server-side Web Applications (Blazor)

Blazor is a framework for building web applications using C# instead of JavaScript. In server-side Blazor, all the application logic runs on the server, and changes in the interface are sent to the browser in real time using SignalR, a communication technology that keeps the client and server synchronized. This allows developers to write full-featured web applications without having to manage client-side code and APIs, while keeping the application state consistent and secure on the server.

2.10 HTTPS and Reverse Proxy

HTTPS (Hypertext Transfer Protocol Secure) is a protocol that encrypts all data sent between a user's browser and the server. This prevents attackers from intercepting sensitive information, such as voter tokens or election results. A reverse proxy is a server that sits between the internet and the application server, routing requests securely and efficiently. It can also help balance traffic and provide an extra layer of protection against attacks.

2.11 Token-based Authentication

Tokens are unique codes used to verify a user's identity and permissions without requiring a username and password. In this system, each voter receives a token that allows them to vote exactly once and ensures anonymity, while administrators receive a separate token to perform sensitive actions, such as ending elections or retrieving results. This approach is simple for users and secure for the system, preventing unauthorized access.

2.12 Email Notifications

Automated emails are used to communicate with voters. Each email contains a unique link with the voter's token so they can access the correct election securely. The system can generate these emails dynamically and send them individually or in bulk from the admin panel, making it easy for organizers to invite voters and send reminders while ensuring that each voter receives only their authorized link.

3 Related Work

The system developed provides formal verifications of both the Borda Count and Single Transferable Vote algorithms, enabling their use as part of a trustworthy electronic voting service.

3.1 Borda Count

The first algorithm implemented and verified is the Borda Count algorithm with an additional tiebreaker representing a modification of Baldwin's method. A study regarding the "Complexity of and Algorithms for Manipulation of Borda, Nanson's, and Baldwin's Voting Rules"[7] has proven the NP-hardness of an attempt to manipulate elections evaluated using Borda Count, Baldwin's, or Nanson's method. In addition, it has been shown that the manipulation for Baldwin's and Nanson's methods requires greater computational effort than for the Borda Count. This provides further motivation for selecting Baldwin's method as a tie-breaking alternative, as its higher resistance to manipulation contributes to election robustness.

Another important reason for selecting Baldwin's method was the preservability of key Borda Count properties. The study "Disagreement of Counting Methods in Simulated Ranked Preference Elections"[5] identifies several features shared by Borda and Baldwin, including the selection of broadly acceptable candidates, utilization of voters' complete preference lists, high similarity in winner selection outcomes across simulations, as well as common violation of the Independence of Irrelevant Alternatives criterion, meaning that the relative relationship between two candidates may be affected by the presence or absence of additional candidates.

Therefore, the Baldwin's method was considered a suitable tie-breaking mechanism, with some modifications. The full Baldwin method involves iterative elimination of candidates with the lowest Borda scores until a single winner remains, whereas the Borda Count determines a winner in a single step when no ties exist among the top candidates. To reduce computational complexity, the modification employed here classifies all uniquely ranked winners at each iteration, thereby reducing the number of elimination rounds required to build a ranking

of a given size.

While related studies such as "Voting Procedures: A Summary Analysis"[13] analyze Nanson's method – which eliminates all candidates scoring less than the average Borda count in each round and satisfies the Condorcet criterion by always selecting the Condorcet winner if one exists – none discuss the modification of Baldwin's method used here. Nevertheless, this design choice balances computational efficiency with preservation of the key properties of the Borda Count, making it suitable for formal verification and trustworthy e-voting applications.

Proceeding to the formal verification of the Borda Count with the proposed tie-breaking mechanism, an analysis of its axiomatic properties was required. "An axiomatic study of the Borda rule on top-truncated preferences" [15] identifies core properties of the Borda Count, such as anonymity (permuting voters does not change the outcome), neutrality (permuting candidate labels results in a correspondingly permuted outcome), reinforcement (if two disjoint electorates yield the same ranking, their union also yields this ranking), monotonicity (raising a candidate's position in any ballot cannot harm them), and cancellation (a perfectly balanced profile yields a complete tie). Additionally, in the case of top-truncated ballots allowing voters to submit incomplete rankings, top-cancellation (if every candidate appears equally often at the top positions, the outcome is a tie), top-equality (candidates occurring identically across all voters' top lists should be tied), and top-consistency with candidate labeling (the outcome depends only on the information present in the top parts of ballots, up to candidate relabeling).

In this setting, ballots may be top-truncated, such that the last three axioms are included into consideration of key Borda count properties. However, since cancellation, top-cancellation, and top- equality enforce tied outcomes, these axioms are violated by the tie-breaking method used, resolving all existing ties in the first k desired places. Nevertheless, anonymity, neutrality, and monotonicity remain preserved, as the modification does not privilege specific voters or candidates and does not penalize candidates for being ranked higher. Reinforcement is preserved in the sense that combining electorates that yield the same strict ranking will still yield that ranking, yet only insofar as no irresolvable ties remain after applying the tie-breaking method, which enforces randomness as a resolver.

Monotonicity was explicitly verified in Dafny, while treating candidates and voters abstractly implies the satisfaction of anonymity and neutrality. Moreover, top-consistency with candidate labeling is preserved due to resolving only the ties in the first k places, without paying attention to candidates' labels. In that way, this can be documented as a conscious trade-off: the tie-breaking method preserves Borda's use of full preference information and its empirical winner profile while providing efficient tie resolution.

Lastly, the formal verification of the modified Borda Count method has taken place in Dafny. Although the Borda Count is one of the most prominent and widely analyzed voting rules, surprisingly little prior work addresses its formal verification. While studies such as "Formalization and Verification of Proportional Representation Voting Systems using Isabelle/HOL"[14] and "Voting Theory in the Lean Theorem Prover"[11] verify key properties of voting systems using interactive proof assistants – Isabelle/HOL based on higher-order logic with strong SAT/SMT integration and Lean based on dependent type theory with tactic-driven proves – none concern the formal verification of the Borda Count algorithm. In contrast, this work employs Dafny, an automated program verifier, to formally verify both correctness and key axiomatic properties. Furthermore, no prior studies address the use of a modified Baldwin's method as a tie-breaking mechanism, highlighting the novelty and contribution of this project.

3.2 Single Transferable Vote (STV)

The second algorithm implemented and verified is a Single Transferable Vote algorithm, specifically the Gregory method. Single Transferable Vote (STV) is a family of vote counting schemes which all consist of the same core mechanisms: transfer, count, elect, eliminate. This imprecise definition captures a wide range of STV counting schemes that are used worldwide in different elections where each election uses some variation of these core mechanisms. The conference paper "Modular Formalisation and Verification of STV Algorithms"[10] offers a framework for proving the correctness of these varying schemes. It introduces so called "sanity checks" for each of the four core routines, defined as post- and preconditions that are precise enough to argue about the whole voting algorithm but loose enough to be able to argue about different variations of the subroutines. During our verification process where able to use these "sanity checks" as rough guidelines, but mostly used more precise contracts to rigorously validate our specific implementation.

The key challenge in automatically verifying voting schemes is selecting voting criteria and arguing about fairness. But no voting scheme satisfies all reasonable general voting criteria. For preferential voting, such as STV, Arrow's impossibility theorem[2] even states that no scheme can be designed to satisfy the three fairness criteria: Unanimity, Independence of Irrelevant Alternatives, Non-dictatorship. This impossibility makes it more practical to evaluate a voting scheme against specific, relevant properties rather than a general standard. Following this logic, the paper "Verifying voting schemes"[3] advocates for developing tailor-made criteria. The paper suggests these two specific criteria for preferential voting:

1. There must be enough votes for each elected candidate
2. If the preferences of all voters w.r.t. two particular candidates are consistent, then that collective preference is not contradicted by the election result.

After converting these criteria into first-order logic formulae, the Z3 SMT-solver is used to verify that Standard Single Transferable Vote preservers these properties. In contrast the CADE-STV election scheme, which is used by the he Conference on Automated Deduction(CADE) to elect members to its Board of trustees, doesn't satisfy the first criterion. The "standard" STV-method being verified in this paper allows some degree of randomness. One could argue that our implementation of the Gregory-method is more fair, because it is fully deterministic and doesn't rely on any randomness.

While the formal verification of STV algorithms is not a new endeavor, ours is, to the best of our knowledge, the first to analyze the Gregory method or any STV variant within the Dafny framework.

4 Technical Sections

This section is split up into two parts. The first part is about the Dafny core of our system, the part where the actual verified voting is happening. The second part is about our Website that offers the voting service to the public.

4.1 Dafny

Approach. Voting is one of the most fundamental ways to address societal decisions. Therefore, the significance of voting systems has grown over time, increasing the demand for trustworthy electronic voting services. However, few existing voting services can guarantee the formal correctness of the applied voting rules and the derived outcomes. With this motivation, the goal of this project is to enable the use of formally verified electronic voting services, ensuring correctness, predictability, and fairness.

This approach addresses two perspectives: the organizer, who defines the voting session, and the voter, who seeks confidence in the security and correctness of the outcome. Additionally, offering a variety of voting rules offered is crucial, since each voting method satisfies some – but never all – democratic properties.[2]

Two algorithms were implemented and verified in Dafny: Borda Count and STV. Both require ranked lists of registered candidates of arbitrary length; however, for STV, votes cannot be empty. In addition, Borda Count calculates points for each candidate based on full preference lists, while STV counts first-choice votes and iteratively redistributes surplus or unassigned votes according to the Droop quota.

Therefore, these voting rules satisfy complementary properties: Borda Count ensures monotonicity but may violate proportionality and the Condorcet criterion, whereas STV allows proportional outcomes but is not monotonic. Offering both enhances the platform's transparency, robustness, and flexibility.

To address ties in Borda Count, a tie-breaking mechanism based on a modified Baldwin's method was designed. In each iteration, the tiebreaker eliminates the candidate with the fewest points, while classifying all uniquely ranked winners to reduce the computational complexity of this iterative approach.

Finally, the formal verification in Dafny ensures that these algorithms behave according to their specifications, without privileging any voter or candidate. Dafny was chosen because it combines implementation and verification in one environment, enables compilation of the written code into C#, supports modular reasoning, and integrates SAT/SMT solving through tools such as Z3.

In this context, key axiomatic properties were targeted. For Borda Count with the proposed tie-breaking mechanism, anonymity (permuting voters does not change the outcome), neutrality (permuting candidate labels results in a correspondingly permuted outcome), and monotonicity (raising a candidate's position cannot harm them) were preserved. Concurrently, trade-offs were consciously made regarding axioms such as cancellation, top-cancellation and top-equality, which inherently enforce tied outcomes.

For STV, verification focused on its quota-based proportionality, anonymity and neutrality, as well as correctness of the iterative transfer process.

These design choices created the foundation for the verified implementation, which followed a compositional structure to ensure scalability and reusability of system components.

Implementation. Building on this foundation, the implementation followed a compositional design of the system, allowing scalable verification and re-use. Each verification process began with the construction of a design plan in pseudocode, breaking the system into manageable components. To validate these design decisions, lightweight prototypes were first created in Python, providing early feedback on correctness before moving to Dafny verification. This incremental approach enabled efficient team collaboration and facilitated formal proofs.

Core data structures in Dafny were determined early on: candidates were represented as sequences of natural numbers, votes as two-dimensional sequences (preference lists of varying lengths), and rankings as mappings from candidates to the number of earned points. These abstractions supported both Borda Count and STV.

For Borda Count, methods were implemented to generate rankings, eliminate candidates, handle tie- breaking, and manage ordering operations such as sorting. The STV architecture reused much of this structure, with additional methods for proportional redistribution of surplus votes, eliminating candidates and

empty votes, and determining redistribution reasons. For both rules, wrapper methods coordinated the sub-methods to ensure algorithms were executed correctly and verifiably in Dafny.
However, for STV an additional restriction was imposed: no empty votes (i.e. empty inner sequences in the two-dimensional sequence of registered votes) were allowed. This decision was motivated by the fact that empty votes could influence the Droop quota and thereby alter the resulting outcome in an undesirable or unrealistic way.

Before verification, a small set of global preconditions and postconditions was defined: duplicate votes were disallowed, and all elements of each vote had to be registered candidates. Postconditions enforced the axiomatic properties of each voting rule. For Borda Count, monotonicity was verified as part of the classification of unique winners. For STV, redistribution of surplus votes – or all votes when no candidate met the quota – was verified through a dedicated method that computed and applied proportional transfer factors.

The most important aspect of the compositional and synchronous implementation and verification was the careful selection of appropriate pre- and postconditions. Since a wrapper method (i.e. a method implementing an entire voting rule by invoking multiple sub-methods) relies on numerous contracts, the strength or weakness of individual specifications became crucial. A precondition that was too restrictive or a postcondition that was too weak could compromise the overall verification, even if each sub- method had already been proven formally correct. These integration steps turned out to be the most challenging part of the project, particularly in the verification of the Borda Count tie-breaking mechanism.

The final and most technically demanding step was verifying the SortCandidatesHelper method, responsible for tie-breaking. This method integrates numerous sub-methods, each with its own pre and postconditions, and was originally intended to be implemented recursively. A major challenge in a team-based verification project was aligning contracts across these sub-methods: frequent amendments were required to ensure the postconditions of one method satisfied the preconditions of the next, often necessitating re-verification of earlier components.

Ultimately, the recursive implementation was replaced by an iterative one using a while-loop, which simplified reasoning and improved verification feedback times. However, the verifier still struggled because the contracts for the sub-methods called within the loop were too complex, causing long verification times. To resolve this, a structural refactoring strategy was employed: instead of invoking the sub methods directly from the while-loop, those calls were factored into simple, linear helper methods, which reused the loop's invariants as their own pre- and postconditions. This separation allowed the main loop to verify quickly, leaving the more detailed verification to simpler, loop-free helper methods.

Finally, the verified code was compiled to C#: all Dafny data structures were mapped to immutable C# data structures, such that, for example, rankings represented as mappings in Dafny became immutable maps in C#. This required minor adaptations at the interface level but did not represent a significant workload.

With the core methods implemented and formally verified in Dafny, the system was ready for evaluation on representative election scenarios.

Evaluation and Testing. After the implementation phase, the system was evaluated against the defined goals. The formal verification confirmed that both voting algorithms satisfied their targeted axiomatic properties: Borda Count with respect to monotonicity and STV with respect to proportionality. In addition, complementary test cases in C# evaluated anonymity and neutrality, demonstrating consistent preservation of these fundamental properties.

To support early validation, both algorithms were first prototyped in Python. Unit tests were used to assess correctness against the pseudo-code and expected behavior. For Borda Count, the focus was on validating sub- and wrapper methods as well as achieving sufficient statement coverage, particularly for the tie-breaking mechanism. In contrast, the iterative nature of STV required tests to target properties of the vote redistribution process, ensuring that redistribution occurred consistently and correctly.

Beyond Dafny verification and Python unit tests, the final C# implementation was applied to representative election scenarios. These tests confirmed that permuting voter or candidate labels yielded outcomes consistent with the principles of anonymity and neutrality. Combining formal verification with empirical testing provided robust evidence of correctness and reliability.

In conclusion, the evaluation confirmed that the verified implementation achieved its targeted axiomatic guarantees and behaved correctly under realistic test conditions. These results provide a foundation for reflecting on the challenges encountered while using Dafny, as discussed in the following section.

Discussion. Working with an unfamiliar verification tool inevitably requires overcoming challenges related to its novelty and internal design. In this project, the main difficulties arose from learning how Dafny verifies properties and how it internally handles different data structures. These challenges shaped the development process and ultimately deepened the practical understanding of formal verification.

A key challenge was recognizing Dafny's limitations when reasoning about certain data structures. Each type comes with properties that Dafny or its underlying Z3 solver does not verify easily. For sequences in particular, verification often failed to terminate unless additional guidance was provided. For example, when splitting a sequence at a valid index i, Dafny does not automatically recognize that the subsequence of the first $i + 1$ elements is equivalent to the subsequence of the first i elements concatenated with the ith element. Similarly, reasoning about concatenation may require explicit assertions to establish that a given element is contained in the resulting sequence.

Overcoming these issues required developing strategies to work around Dafny's weak spots and to provide precise assertions where needed. Identifying such problem areas and learning how to address them significantly improved our ability to use Dafny effectively and provided valuable insights into practical realities of formal verification.

Another major obstacle was the integration of sub-methods into wrapper methods. The strength or weakness of preconditions and postconditions became crucial, requiring careful contract specification to ensure compatibility between the components. On the one hand, modular implementation simplified the verification of individual sub-methods by reducing their complexity. On the other hand, composition introduced new challenges, since the correctness of the overall algorithm depended on the precise interaction of all specifications. This trade-off was consciously accepted, as modularity ultimately provided better structure and re-usability, despite the added complexity during integration.

An additional issue arose from two different definitions for the "uniqueness of elements in a sequence" well condition. One team member defined it using pairwise comparisons, while another used Dafny's multiset properties. Dafny could not automatically prove that these definitions were logically equivalent, so a specific lemma was required to connect them. This highlighted the importance of consistent definitions in collaborative verification projects.

In summary, this discussion reveals that working with Dafny requires not only technical precision but also adaptability in developing proof strategies. The difficulties encountered – ranging from sequence reasoning to contract integration – served as opportunities to better understand the interaction between specification and implementation. Ultimately, the project demonstrated that formal verification is not only a technical process but also an iterative learning experience, where tool limitations, specification clarity, and collaboration play equally important roles.

4.2 Server

Approach. While Dafny was used to formally verify the correctness of the voting rules, a reliable and scalable server infrastructure was required to make these algorithms accessible to real users. The server needed to fulfill three core goals:

1. Provide a user-friendly interface for organizers and voters.
2. Integrate the formally verified C# voting library compiled from Dafny.
3. Persist election-related data securely and consistently.

To achieve this, a server-side Blazor application using .NET was chosen. This allowed building an interactive web application with multiple server components while retaining a single, consistent technology stack in C#. The architecture was designed around a clear separation of concerns: the database manages persistent election data, the Dafny libraries manage the evaluation of election votes, and the Blazor UI handles user interactions.

The database schema was deliberately kept minimal to ensure clarity, consisting of four main tables:

1. **election:** general information about each voting session with a specified voting algorithm
2. **voter:** registered voters linked to an election and identified by their email address
3. **vote:** anonymized ballots submitted by voters
4. **candidate:** a possible voter option linked to an election

This schema enforces anonymity by separating voter identities from cast votes, while still allowing election organizers to validate participation. During development, a MySQL database was used, with deployment planned on MariaDB for production environments.

Implementation. The server implementation was designed to provide a secure and trustworthy environment for elections, guiding users through processes like election creation, casting votes or viewing election results while ensuring transparency and integrity at every step.

Organizers use the /create page to set up a new election. They provide a name and description, select a voting system, set the end date, and configure privacy options. Candidates and voters are added interactively with input validation. Once submitted, the election is stored in the database, and a unique administrator token is generated, giving the organizer secure access to manage the election.

Voters access the election via /vote/{token}. The server first redirects the user to a page that matches the election's voting system for example, a Borda Count election leads to a preference list input page. The interface presents the list

of candidates and allows voters to assign scores or rankings according to the election type. All input is validated in real time to enforce election rules, such as unique rankings or valid score ranges. Once submitted, votes are anonymized and stored in the database, and the voter is marked as having voted. The page is providing immediate feedback and confirmation dialogs, which help voters trust that their actions are recorded correctly and have a direct consequence.

Administrators access elections using a unique admin token /admin/{admintoken}. The panel displays election metadata, voter progress, and the list of registered voters. Administrators can update the election name and description, resend voter invitations, or manage the election life cycle. Critical actions, such as ending the election early or deleting it, are protected by confirmation dialogs to prevent accidental changes. For public elections, a QR code linking to the voting page is provided for easy distribution. The panel integrates the verified Dafny library for computing results, ensuring that outcome calculations remain correct and auditable

To ensure timely and correct processing of completed elections, a background thread continuously monitors the database at regular intervals. When it detects an election whose end conditions have been met, it automatically triggers result computation using the verified Dafny DLL via the wrapper class. Once the results are computed, the system updates the database and, if applicable, sends email notifications to voters indicating that the election has concluded. This automated workflow guarantees that all completed elections are evaluated reliably, leverages the formally verified voting logic, and maintains transparency by notifying participants promptly.

Several technical design choices were made to ensure scalability, security, and maintainability of the server:

- **Stateless database access:** A static database class provides centralized, stateless access to all tables. This approach ensures consistent data access across multiple Blazor components.
- **Token-based authentication:** Unique tokens for voters and administrators enforce access control. Each voter token guarantees one-time participation and anonymity, while admin tokens restrict sensitive operations such as ending or deleting elections.
- **Secure transmission:** The site is hosted over HTTPS via a reverse proxy, ensuring that tokens and other sensitive data cannot be intercepted during transmission.
- **Email notifications:** A dedicated mail class handles sending emails to voters, including invitations and reminders. Voters receive a link to their election via email containing their unique token, but there is also an option on the website to manually enter the token if needed. Emails can be generated dynamically and resent individually or in bulk from the admin panel.

These design choices collectively provide a robust and auditable infrastructure, allowing elections to run reliably while maintaining the integrity and verifiability of all interactions.

Evaluation and Testing. For the Dafny core algorithms, unit tests were implemented directly on the server using sample elections and vote distributions. For example, Borda Count was tested with multiple permutations of votes to confirm anonymity and consistency, ensuring that reordering votes does not change the outcome. Additional tests validated edge cases, such as improperly formatted votes (e.g., including non-numeric entries) and votes of varying lengths, to verify that errors are handled gracefully and results remain correct. These tests directly call the Dafny wrapper, deserialize the output, and compare it against expected results.

The actual server code was also tested extensively, including workflows for election creation, voting, and administrative operations. Validation is enforced both in the user interface and in the database. This includes constraints to prevent illegal states, such as duplicate voters, invalid candidate IDs, or votes cast outside of the election period. The database enforces referential integrity between elections, candidates, voters, and votes, while the server logic validates all user input before committing changes. Together, these safeguards ensure that the system remains consistent, robust, and resistant to both accidental errors and malicious attempts to manipulate the election data, maintaining trustworthiness throughout the process.

Discussion. Developing the server involved several challenges. Ensuring votes were securely transmitted and stored required implementing token-based authentication, HTTPS hosting, and careful database constraints to prevent illegal states. Managing real-time interactions with server-side Blazor and SignalR was complex, especially to handle concurrent vote submissions without inconsistencies. Integrating the Dafny wrapper demanded careful serialization and error handling to ensure formally verified algorithms worked correctly in the live system. Finally, automating election evaluation and notifications via a background thread required coordination with the database and safeguards to guarantee results were computed exactly once. These challenges highlighted the need for careful design, rigorous testing, and modular architecture to maintain security, correctness, and trustworthiness.

5 Project Management

Our team followed a collaborative and egalitarian approach to project management. All members were considered equal, and major decisions were always made collectively to ensure that every perspective was considered.

We held a weekly team meeting, where we discussed the progress made during the past week, evaluated any difficulties that had arisen, and defined concrete goals for the following week. These meetings provided structure and allowed us to continuously monitor the overall development of the project.

In addition to the scheduled meetings, we frequently collaborated in smaller groups or pairs, often meeting in person between courses to address specific technical questions or implementation details. Since all team members study computer science at the same university, in-person communication was our primary mode of collaboration and proved to be both effective and efficient.

When physical meetings were not possible, we made use of Zoom calls for discussions and relied on messenger services for quick communication. This combination of structured weekly meetings and small, flexible consultations allowed us to stay aligned and maintain steady progress throughout the project.

5.1 Contributions of the Individual Team Members

Büttner, Robert. At the beginning of the project, Robert worked on the initial Python prototype of the Borda Count algorithm, which had been designed by Henrik Oback, and extended it into the version used for the first demonstration. Following this, he contributed to the implementation and formal verification of both voting algorithms in Dafny, collaborating closely with Ivana Kostadinovic and Patrick Janoschek. During this phase, he also supported coordination within the team, helping to align individual contributions and ensure that the different components integrated smoothly. In addition, he represented the team externally by giving presentations about the project.

Janoschek, Patrick. Early in the Project, Patrick provided the first formal description of the Borda Count algorithm in pseudo-code, which served as a starting point for further development of the algorithm. He also developed a comprehensive requirements analysis and a set of use case scenarios for the project. Researching Dafny before the verification work had started, naturally led to his main Focus: the formal verification of the voting algorithms, a collaborative effort with Robert Büttner and Ivana Kostadinovic. Alongside this, he implemented a CI/CD pipeline in GitLab to automatically check the verification of each Dafny file upon each commit, streamlining the development workflow.

Kostadinovic, Ivana. At the beginning of the project, Ivana was responsible for handling edge cases in the Borda Count, including the design of the tie-breaking mechanism. Therefore, she refined the initial pseudo-code and extended the requirements analysis, providing a foundation for later implementation. In the development phase, she contributed to a Python prototype to validate the pseudo-code and implemented unit tests with emphasis on statement coverage, ensuring correct handling of tie-breaking logic. Furthermore, she developed the

prototype for the Single Transferable Vote (STV) and carried out testing of half of its methods against the required properties. In the final phase, Ivana, together with Robert Büttner and Patrick Janoschek, contributed to the formal verification of both the Borda Count and STV using Dafny.

Oback, Henrik. During early development of the project, Henrik acted as the team leader, guiding discussions, assigning tasks, and coordinating the development plan. He also implemented an initial Python prototype of the Borda Count algorithm and set up the Git infrastructure to support collaborative development. In the second half of the project, leadership responsibilities were taken over by Robert Büttner, allowing Henrik to focus entirely on server development. He independently designed and developed the full server infrastructure, including the database, backend logic, and frontend Blazor pages, handling all aspects of election creation, voting, administration, and integration with the verified Dafny algorithms.

6 Conclusions and Future Work

Our project successfully met its primary goal of designing and implementing a web-based e-voting system built upon a formally verified core using the Dafny programming language. In doing so, we have provided an effective method for bridging the gap between the theoretical formal methods and the real-world need for trustworthy digital systems.
A review of the final product demonstrates a successful fulfillment of our stated project goals. We formally verified two distinct and complex preferential voting algorithms in Dafny: Borda Count, with a modified Baldwin's method for tie-breaking, and Single Transferable Vote (STV), using Gregory's method. This verified core was then successfully embedded within a secure and accessible Blazor web application that ensures voter anonymity and provides a user-friendly interface for setting up both public and private elections. Finally, to ensure transparency, the complete Dafny source code was made publicly available for independent inspection.
This project serves as a modular foundation that opens up many possibilities for further development. Future work could expand the selection of verified voting algorithms to include algorithms, such as the simpler Approval Voting or the more complex Condorcet Method. A more significant advancement would be to implement cryptographic End-to-End (E2E) verifiability, which would allow voters to confirm their ballots were counted correctly without compromising the voters privacy. Since our verification only applies to the correctness of the algorithms themselves and not the entire software and hardware stack on which the application is deployed, the scope of formal verification could be broadened to encompass more of the system, further strengthening the chain of trust.

References

1. Alvarez, R.M., Hall, T.E.: Electronic Elections: The Perils and Promises of Digital Democracy. Princeton University Press (2010)
2. Arrow, K.J.: A difficulty in the concept of social welfare. Journal of Political Economy **58**(4), 328–328 (None 1950). https://doi.org/10.1086/256963, https://ideas.repec.org/a/ucp/jpolec/v58y1950p328.html
3. Beckert, B., Goré, R., Schürmann, C., Bormer, T., Wang, J.: Verifying voting schemes. Journal of Information Security and Applications **19**(2), 115–129 (2014). https://doi.org/https://doi.org/10.1016/j.jisa.2014.04.005, https://www.sciencedirect.com/science/article/pii/S2214212614000246
4. Black, D.: The Theory of Committees and Elections. Cambridge University Press (1958)
5. Calia, B., Sivek, J.: Disagreement of counting methods in simulated ranked preference elections (oct 2023), unpublished manuscript
6. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)
7. Davies, J., Katsirelos, G., Narodytska, N., Walsh, T., Xia, L.: Complexity of and algorithms for the manipulation of borda, nanson's and baldwin's voting rules. Artificial Intelligence **217**, 20–42 (2014). https://doi.org/https://doi.org/10.1016/j.artint.2014.07.005, https://www.sciencedirect.com/science/article/pii/S0004370214000885
8. Emerson, P.: Designing an All-Inclusive Democracy: Consensual Voting Procedures for Use in Parliaments, Councils and Committees. Springer (2013)
9. Farrell, D.M., McAllister, I.: The Australian Electoral System: Origins, Variations and Consequences. UNSW Press (2006)
10. Ghale, M.K., Goré, R., Pattinson, D., Tiwari, M.: Modular formalisation and verification of stv algorithms. In: Krimmer, R., Volkamer, M., Cortier, V., Goré, R., Hapsara, M., Serdült, U., Duenas-Cid, D. (eds.) Electronic Voting. pp. 51–66. Springer International Publishing, Cham (2018)
11. Holliday, W.H., Norman, C., Pacuit, E.: Voting theory in the lean theorem prover (2021), https://arxiv.org/abs/2110.08453
12. Leino, K.R.M.: Dafny: An automatic program verifier for functional correctness. In: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-16). pp. 348–370. Springer (2010)
13. Nurmi, H.: Voting procedures: A summary analysis. British Journal of Political Science **13**(2), 181–208 (1983). https://doi.org/10.1017/S0007123400003215
14. Rossetta, S.F.: Formalization and Verification of Proportional Representation Voting Systems using Isabelle/HOL: A Study of D'Hondt and Sainte-Laguë Methods. Master's thesis, Politecnico di Torino (2024), corso di laurea magistrale in Ingegneria Informatica (Computer Engineering)
15. Terzopoulou, Z., Endriss, U.: The borda class: An axiomatic study of the borda rule on top-truncated preferences. Journal of Mathematical Economics **92**, 31–40 (2021). https://doi.org/https://doi.org/10.1016/j.jmateco.2020.11.001, https://www.sciencedirect.com/science/article/pii/S0304406820301208
16. Tideman, T.N.: The single transferable vote. Journal of Economic Perspectives **9**(1), 27–38 (1995)