Modeling and Analyzing Faults to Improve Election Process Robustness

Borislava I. Simidchieva
UMass Amherst [*]
bis

Sophie J. Engle
UC Davis [†]
sjengle

Michael Clifford
UC Davis
mclifford

Alicia Clay Jones
Booz Allen [‡]
jones_alicia

Sean Peisert
UC Davis/LBNL
speisert

Matt Bishop
UC Davis
mabishop

Lori A. Clarke
UMass Amherst
clarke

Leon J. Osterweil
UMass Amherst
ljo

Abstract

This paper presents an approach for continuous process improvement and illustrates its application to improving the robustness of election processes. In this approach, the Little-JIL process definition language is used to create a precise and detailed model of an election process. Given this process model and a potential undesirable event, or hazard, a fault tree is automatically derived. Fault tree analysis is then used to automatically identify combinations of failures that might allow the selected potential hazard to occur. Once these combinations have been identified, we iteratively improve the process model to increase the robustness of the election process against those combinations that seem the most likely to occur.

We demonstrate this approach for the Yolo County election process. We focus our analysis on the ballot counting process and what happens when a discrepancy is found during the count. We identify two single points of failure (SPFs) in this process and propose process modifications that we then show remove these SPFs.

1 Introduction

An election is the "formal choosing of a person for an office, dignity, or position of any kind; usually by the votes of a constituent body" [49]. Such a choosing requires a process, or a sequence of actions, that result in a selection. This process may be as simple as counting raised hands in a room, or as complex as tallying votes across a multiplicity of jurisdictions, each of which uses its own rules to make the votes available.

Indeed, part of the process is determining *which* votes to count, and who is a member of the "constituent body." The rules governing this are essentially a political issue, because different rules apply to different types of elections. An equally important part of the election process is validating the results, to confirm that the votes were counted correctly. Here, the degree of certainty in the correctness of the result and the selection of the method used to do the validation are both political questions, however, how the selected method is implemented is not a political question, but rather a technical one.

The process is important because the results of an election can affect the course of history. Imagine how different United States history would have been had George McClellan become president in 1864 rather than Abraham Lincoln. Had Lyndon Baines Johnson lost his first Senate race, rather than winning by 87 votes, much of the civil rights legislation of the 1960s may never have been passed. History is replete with examples of where elections changed history.

An election process involves people. The election officials, candidates, poll workers, voters, and election judges all participate directly in the process. Less directly, legislators, judges, regulators, and others who set the rules for who can vote, and how the elections are to be conducted, also participate. The rules may be complicated, especially when a single ballot includes races from multiple jurisdictions, each with its own rules. A good example is a ballot for an election for federal, state, and local candidates in San Francisco, CA. For some local races, San Francisco uses ranked-choice voting[1]; but for all state and federal candidates, the rules require majority voting. Thus, the votes for two different races on the same ballot are counted differently.

Less obvious, but no less serious, problems often arise in the voting process. For example, a ballot box may not be turned in when required, or a set of votes may be counted twice. These problems can be detected

[*]Laboratory for Advanced Software Engineering Research (LASER), Department of Computer Science, University of Massachusetts Amherst. For emails add @cs.umass.edu

[†]Computer Security Laboratory, Department of Computer Science, University of California, Davis. For emails add @ucdavis.edu

[‡]Booz Allen Hamilton. For email add @bah.com

[1]Ranked-choice voting effects instant runoff when no candidate receives more than 50% of the first-choice votes. For details, see http://www.sfgov2.org/index.aspx?page=876

quickly when experienced election officials have anticipated them, and have made appropriate provisions for their detection and correction. Although most election districts have enacted provisions for handling known problems, unexpected issues may still arise. Thus, election processes must be constantly improved to address contingencies.

Currently, election officials use *ad hoc* approaches both to address problems as they arise and to anticipate problems before they arise. We advocate the use of *continuous process improvement technology* to complement existing ad hoc approaches. In our work we formalize election processes and desired properties. We then use various analysis approaches to identify potential problems that might occur, thus systematizing the search for problems before they arise. Once problems have been identified, either in this anticipatory fashion, or through actual experience in an election, we use these same forms of analysis to confirm that process modifications successfully address the problems. Typically, modifications entail adding new checks and redundancies into the processes so that multiple simultaneous failures are necessary to cause a desired election property to be violated. We do not distinguish between accidental failures and failures that an adversary deliberately causes (attacks). Both can be equally pernicious.

Election process details vary among jurisdictions. To demonstrate a specific, real-world application of our techniques, we focus on the election process used by Yolo County, California[2]. However, our technique is generalizable, and is equally relevant and applicable to many other jurisdictions' election processes. The first step in our approach is to develop a precise and rigorously defined model of the election process. This paper concentrates specifically on the part of the Yolo County process that deals with the counting of votes. The next step is to apply a variety of analyses to this process model, in order to identify various types of process problems. This paper focuses on one such analysis, namely Fault Tree Analysis (FTA). We demonstrate how we derive fault trees from the Yolo County process model, and how we use the fault trees to derive *Minimal Cut Sets* that specify which combinations of events cause election process failures. After suggested process modifications are identified with the help of election officials, we show how repeated application of our analysis technologies can assure that such proposed changes successfully eliminate flaws in the process, leading to continuous process improvement.

The rest of this paper is organized as follows. Section 2 provides an overview of related work. Section 3 introduces our approach and details how it can support continuous process improvement. In Section 4, a case study of process faults identified by Yolo County election officials is presented, and a discussion of how our approach can handle such faults is included. Finally, Section 5 presents a conclusion and some directions for future research.

[2]http://www.yoloelections.org/

2 Related Work

This paper applies process modeling and fault tree analysis to the area of elections. In this section, we examine work related to these areas of research.

2.1 Electronic Voting and Election Requirements

A very active community is investigating approaches to improving electronic voting systems [6, 30, 44]. Work in this area is focused primarily on the security of the devices used for electronic voting. Interest in electronic voting systems is longstanding (such devices have been used since the 1960s), but recent new technology has brought a plethora of new problems. Numerous studies have demonstrated several security-related concerns with electronic voting systems [11, 25, 26, 38, 39, 52]. For example, a study commissioned by the state of California found that these systems were "inadequate to ensure the accuracy and integrity of the election results" [4, 33]. A similar study from the state of Ohio concluded that these systems "failed to adopt, implement and follow industry standard best practices" [8]. The Federal Election Commission (FEC) and the Election Assistance Commission (EAC) developed a series of standards that electronic devices should meet in order to be certified, the latest of which is the 2005 Voluntary Voting System Guidelines (VVSG) [14, 18, 19]. In addition to these standards, significant work has focused on the requirements that an election as a whole must satisfy, such as privacy, anonymity, accessibility, etc. [27, 31]. Accordingly, the EAC continues development of a set of Election Management Guidelines (EMG) to complement the technical standards for voting equipment [15]. These standards and guidelines, however, focus only on the electronic voting system itself. This work does not extend to the analysis of actual election processes, which is the subject of this paper.

2.2 Process Modeling

It has previously been noted that the security of an electronic voting system alone does not provide assurance of the security or accuracy of an election [3]. For example, most elections require that an eligible voter be allowed to vote no more than once. However, this requirement is typically enforced by a process external to the electronic voting system. Studying the effectiveness of such a process in satisfying such a requirement seems to us to fall within the area of process modeling and analysis, which "focuses [on] interacting behaviors among agents, regardless of whether a computer is involved in the transactions" [12].

Raunak et al. apply process modeling and analysis to election processes to determine whether fraudulent behavior can result in incorrect election results [40], and Simidchieva et al. expand this approach to determine whether an election process meets selected requirements [48]. Here we focus on expanding this approach to improve the robustness of election processes using fault tree analysis.

Antonyan et al. study how additional auditing procedures may improve the integrity of elections, and illustrate their approach for AccuVote Optical Scan systems and a generic election procedure [2]. The authors focus on how election procedures affect the ability to prevent or detect attacks to the underlying election systems, whereas our work focuses on how the election procedures themselves may fail. Hall et. al. have also examined audit procedures, specifically focusing on post-election audits [21, 22]. Like our work, the authors examine the procedures for a specific county and use iterative process improvement before generalizing their approach. Our work, however, is not focused on audit procedures. Instead, we focus on automatically finding points of failure in specific election processes.

2.3 Fault Tree Analysis

Fault Tree Analysis (FTA) is a deductive safety and reliability analysis technique that focuses on how a failure state or hazard may occur in a system [17, 50]. A fault tree diagram has some similarity to an electronic circuit diagram, comprising events and logic gates that control whether an event may eventually lead to the hazard at the root of the tree. Fault tree analysis is used by numerous safety-critical industries, including the aerospace, nuclear power, and automotive industries. Brooke et al. demonstrate that in addition to safety-critical systems, fault trees may also be used to analyze security-critical systems [7]. For example, Helmer et al. use software fault trees for intrusion detection systems [23], Zhang et al. use fault trees for vulnerability evaluation [54], and Rushdi and Ba-Rukab apply fault trees to measure a system's exposure to a vulnerability [43]. Yee discusses how safety cases, a construct similar to fault trees, may be used to increase our confidence in voting systems [53].

Attack trees are structurally the same as fault trees, and are used by computer security analysts to model the different paths an attacker may take to reach an objective [45]. Attack trees have been used in penetration testing [29], in identifying insider attacks [41], and for forensics [5, 34, 35, 37]. Nai Fovino et al. combine both fault trees and attack trees for quantitative security risk assessment [32]. Attack trees are also similar to attack graphs [36, 47]. Unlike attack trees, however, attack graphs may be cyclical and do not use logic operators between nodes.

Attack tree analysis generally assumes that faults arise from malicious intent. Since we do not ascribe an intent to how these faults arise, we focus on fault tree analysis in this paper. However, since both fault trees and attack trees are structurally equivalent, this analysis applies to attack trees as well.

3 Approach

Our approach focuses on modeling and analyzing the process used to conduct an election. We use this approach to study how voting systems (which in fact need not be electronic) are configured, used, and checked, and to study other parts of the process such as registering voters and consolidating results from different precincts. Using models of such processes, we are able to apply a range of analyses to infer a variety of characteristics and properties of the modeled election process. Simidchieva et al have applied finite-state verification [48], and Raunak et al have applied discrete event simulation [40] to such models, and this approach extends that work. Analyzing the fitness of specific capabilities (e.g., a specific electronic voting system) for incorporation into a specific election process is a future direction that seems well-supported by our process analysis approach.

In this paper we focus on one specific process analysis technology, namely fault tree analysis. We demonstrate a precise, detailed model of an election process, and then develop fault trees that illustrate potential problems with that process—including those that an attacker can exploit. We then offer suggestions for hardening the process against these problems.

The interested reader is encouraged to download and explore the full election process model (part of which is presented in Section 3.2), full specifications of the fault trees (the top levels of which are outlined in Section 3.4), and additional artifacts and technologies described in this paper from the following website: `http://laser.cs.umass.edu/elections/`

3.1 Process Improvement

The basic tenets of continuous process improvement were introduced by Shewhart [46] and applied with perhaps the greatest effect by Deming [13]. The essence of this approach is to capture the process to be improved, compare its characteristics to those that are desired, identify weaknesses and shortcomings in the process, propose and evaluate improvements, and then install those improvements in the process to complete the improvement cycle and form the basis for a subsequent improvement cycle. This cycle has been referred to in various ways (e.g., the Plan-Do-Check-Act, or PDCA, Cycle and Define-Measure-Analyze-Improve-Control or DMAIC) over the past decades. In all of its names and manifestations it has relied most essentially upon the ability to understand the process, understand its desired properties, and analyze the ways in which the process does or does not adhere to those properties.

Originally, and typically still, these understandings and analyses have been obtained informally. Processes and properties have typically been described in informal natural language, and analyses of their conformance have typically been done with informal discussion and argumentation. More recently, however, research has shown that processes and properties can be defined with precise and rigorous notation, and that doing so then renders the evaluation of their consistency amenable to powerful technological support. Our proposed approach promises to move the venerable informal process improvement approach towards a disciplined engineering practice supported by scientific rigor. Process definitions have also been used as the basis for reasoning about processes in several other domains, including science (e.g., [1, 16]) , medicine (e.g., [10, 24]), and business (e.g., [20, 51]).

3.2 Example Election Process

As noted above for specificity we use an abstract version of the Yolo County election process as the basis for an example of the application of our approach. This example election process model assumes a physical ballot and includes some steps that assume the correct performance of a voting machine and a ballot scanner, but no assumptions are made about the internal workings or the nature of the implementation of the voting machine or scanner technology. The example process consists of three main phases–the pre-polling events that must occur before the polls open on Election Day, the voting events that occur on Election Day, and the counting–and possible recounting–of the votes after the election is over. The first two phases of the process model are specified at a fairly high level and therefore in addition to representing the Yolo County election process as elicited from election officials, the model may also apply to many other jurisdictions' processes. The third phase, namely how ballots are counted, is modeled in more detail in close collaboration with election officials to ensure validity, i.e. that the process model accurately represents the Yolo County election process and does so with sufficient specificity. Because of this specificity, the model includes a counting process based on a central-count optical scan (CCOS) system, but the model is technology-agnostic with respect to specific machines or devices used.

The pre-polling events include training election officials and registering eligible voters. Before the election begins on Election Day, the voting rolls (i.e. the lists of registered voters) are generated and distributed to the precincts. The events that occur on Election Day include the initial checks and setup of any voting machines that may be used and the actual voting process, which encompasses verifying the voter's credentials and allowing the voter to cast a ballot. In the case where the voter is not listed on the voting roll or has been previously checked off as having voted, the voter casts a provisional ballot, which is kept separately from the regular ballots. After the election is complete, the ballot totals at each precinct are recorded onto a summary sheet. This summary indicates how many ballots have been cast or spoiled, and how many were cast provisionally or remain unused, but does not detail any vote totals. The summary sheet and the physical ballots from each precinct are then transported to a higher-level election authority, called Election Central, that performs vote counts, and may also consolidate totals from multiple voting sites, do vote-total verification, and declare official election results. In exceptional situations, such as when the reported ballot totals from a precinct do not match the totals of the ballots received at Election Central, an exception handling subprocess is invoked, which for this case may include either rescanning or recounting the ballots, or both.

3.3 Process Definition Technology

We use the Little-JIL [9] visual process definition language as a vehicle for modeling the Yolo County election process. A Little-JIL process coordination diagram, such as the one shown in Figure 1, consists of a hierarchical decomposition of steps. Note that Little-JIL is a language with rich semantics that allows the precise definition of many different aspects of processes, such as coordination, exception handling, agents, artifacts, and more; the diagrams presented in this paper elide many of these important details to avoid visual clutter–readers wishing to explore the full process model specification should download it from the project website.

A step is denoted by a black bar, with the step name appearing above that bar. Associated with each step is an agent that is responsible for its execution. The behavior of a step consists of the behaviors of its children (the steps that connect to the lower left side of the parent step bar via edges) and the order in which they are executed. Each step that has children also has a sequence badge, which appears in the left half of the step bar and specifies the order in which its children will be executed. For example, the root step `conduct election` in Figure 1 has an arrow, which means it is a sequential step, and hence all of its children are to be executed in left to right order. A step without children is called a leaf step and responsibility for execution of such a step is left entirely to the step's agent. A step is reasonably thought of as a procedure that is invoked whenever there is a reference to that step from anywhere in the process definition.

In addition to the coordination diagram, a Little-JIL process definition includes an artifact specification and an agent specification. The artifact specification consists of all the artifacts that are used in the process, which for this election process example includes `ballot repository` (a repository containing all the ballots cast in one precinct) and `tallies` (a report of the number of votes cast for each candidate). Each step has artifact declarations (not shown in the diagrams presented here) to define which artifacts it will be accessing or providing. Artifacts are generally passed within the coordination hierarchy (from parents to children and vice versa). As steps can be thought of as procedures, this artifact passing is essentially a parameter passing mechanism.

The agent specification indicates for each process step what kind of agent is responsible for its execution. One of the important features of Little-JIL is that it allows the definition of both human and automated (executed by a hardware device or software system) agents. For the election process, `Voter`, `Election Official`, and `Precinct` are some example types of agents. Note that the former two are human agents while the latter can be an automated agent. Little-JIL processes only specify the type of agent (e.g., `Voter`) that should execute a specific step, rather than a specific agent instance (e.g., Jane Doe).

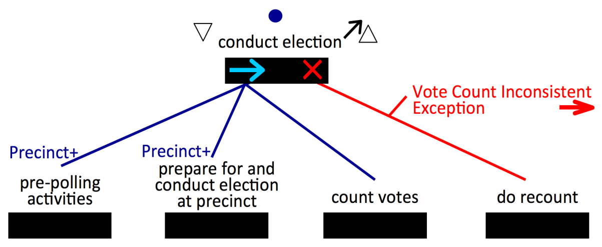

conduct election
Vote Count Inconsistent Exception
Precinct+
Precinct+
pre-polling activities
prepare for and conduct election at precinct
count votes
do recount

Figure 1: The `conduct election` process.

Little-JIL also provides comprehensive exception handling semantics. For example, the `do recount` step in Figure 1 connects to the `X` in the right half of the step bar of its parent, `conduct election`, which indicates that `do recount` is an exception handler. Exceptions in Little-JIL are typed and different exception handlers must be defined for each exception kind. The `do recount` step is an exception handler for exceptions of type `Vote Count Inconsistent Exception` as the edge that connects `do recount` to its parent indicates. Finally, Little-JIL's exception handling mechanism also provides flexible continuation semantics after exception handling takes place. The arrow next to the `Vote Count Inconsistent Exception` notation indicates that after the exception has been handled, the process will continue as if the step that originally raised the exception completed successfully.

As noted, the root step `conduct election` is a sequential step and so all of its children, namely `pre-polling activities`, `prepare for and conduct election at precinct`, and `count votes` are to be executed in this specific order. Note that the edge connecting `conduct election` to `pre-polling activities` has the notation `Precinct+`. The `+` indicates that the `pre-polling activities` step will be instantiated one or more times, once for each agent that executes it. In the case of the `pre-polling activities` step, the agent is specified to be of type `Precinct`, so this step will be executed once for each agent instance of type `Precinct`, which effectively enumerates all Precincts specified as resources, and managed by the Little-JIL resource manager.

After the `pre-polling activities` are completed, `conduct election` proceeds by next executing `prepare for and conduct election at precinct`. Since the edge leading to it again has the `Precinct+` notation, this step is also instantiated multiple times, once for each executing agent, in this case specified to be of type `Precinct`. Once the election at all precincts completes, the execution continues with the `count votes` step, where the vote totals are counted. The `count votes` step can trigger an exception in the case of a vote count discrepancy that cannot be resolved at the precinct level, in which case the `do recount` exception handler of the `conduct election` step is invoked. The substeps that elaborate the `do recount` exception handler, as well as the `pre-polling activities` and `prepare for and conduct election at precinct` subprocesses are omitted here due to space considerations.

The `count votes` subprocess is further elaborated in Figure 2. The `count votes` step is decomposed into the `count votes from all precincts` step, indicating that consolidation of ballots from all participating precincts must occur. Note that once again, the edge to `count votes from all precincts` carries the `Precinct+` notation, indicating that the subprocess will be executed once for every precinct in the system. In turn, `count votes from all precincts` consists of executing `perform ballot count` and `add vote count to vote total` in exactly this order, as pre-

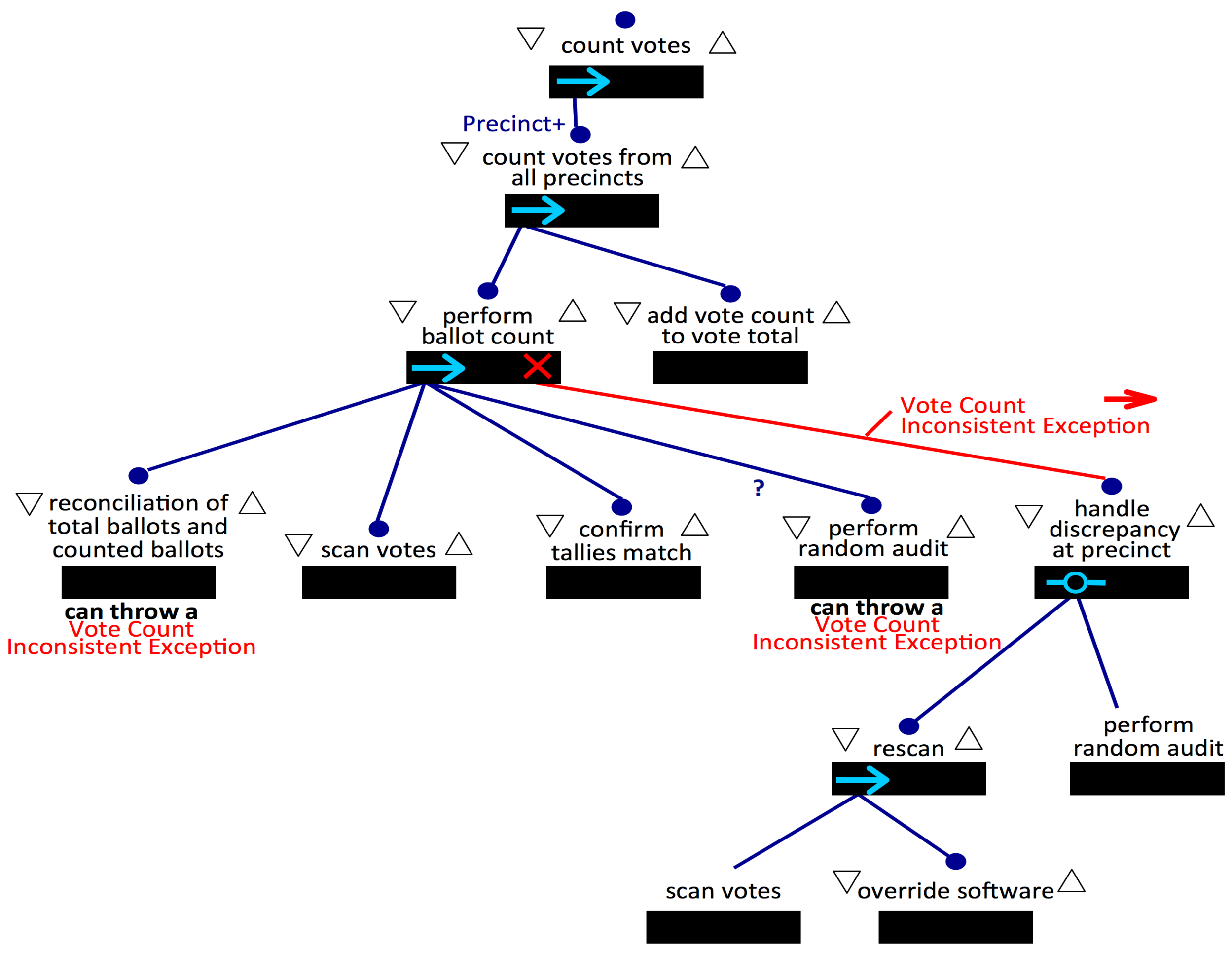

count votes
Precinct+
count votes from all precincts
perform ballot count
add vote count to vote total
Vote Count Inconsistent Exception
reconciliation of total ballots and counted ballots
can throw a Vote Count Inconsistent Exception
scan votes
confirm tallies match
?
perform random audit
can throw a Vote Count Inconsistent Exception
handle discrepancy at precinct
rescan
perform random audit
scan votes
override software

Figure 2: The `count votes` subprocess of `conduct election`.

scribed by the arrow sequence badge of their parent. Effectively, the model states that after the votes from each precinct are counted, the precinct totals are added to the current Election Central totals, and this continues until the votes from all precincts are counted.

The `perform ballot count` is further decomposed into four substeps, executed in left to right order as indicated by the sequence badge of their parent, an arrow. To `perform ballot count`, first a `reconciliation of total ballots and counted ballots` takes place. During this step, election officials confirm that the ballot counts on the summary sheet they have received from the precinct corresponds to the physical ballots they have received. The summary sheet indicates how many total ballots were issued to the precinct, how many of them were used, and how these used ballots break down into cast, spoiled, and provisional ballots. If there is a discrepancy between the ballots received and the ballots reported on the summary sheet, the `reconciliation of total ballots and counted ballots` step can throw a `Vote Count Inconsistent Exception`. This exception would be caught by the `handle discrepancy at precinct` handler as the edge emanating from the `X` on `perform ballot count`'s step bar and leading to the handler indicates.

Next, the election officials `scan votes`, which in the real-world process consists of passing ballots through scanner machines that read the voter intent and keep a running count. As noted, the process model is technology-agnostic, so no assumptions are made in the model about *how* ballots are scanned. After all ballots are scanned, election officials have to `confirm tallies match` by comparing the tallies, or vote counts reported by `scan votes` and ensuring that those numbers are consistent with the ballot counts reported on the precinct's summary sheet. Optionally, after the tallies are confirmed, the election officials may perform a precinct-specific manual count or recount of some or all received ballots (for example in the case where the ballots received do not match the ballots reported on the precinct summary sheet). This step, called `perform random audit` in Figure 2, is denoted as optional by the `?` notation on the edge leading to it. If this step uncovers a discrepancy, it can also throw a `Vote Count Inconsistent Exception`.

The `Vote Count Inconsistent Exception` indicates that a vote count at a given precinct is inconsistent, and can be handled differently depending on the circumstances. The handler step for this exception is `handle discrepancy at precinct`, indicating that if there is any observed discrepancy for the ballots from a given precinct, an attempt will be made to address that discrepancy immediately before the vote counts from this precinct have been added to the running totals. The step `handle discrepancy at precinct` has a circle with a line through it as its sequence badge, indicating that it is a choice step. That means that `handle discrepancy at precinct` can be carried out by choosing to execute exactly one of its chilren–either `rescan` or `perform random audit`–but not both. Thus, depending on the circumstances, election officials may decide that the discrepancy was due to erroneously rescanning a batch of ballots twice, indicating a rescan, or that the discrepancy was due to another reason, indicating an audit. If `rescan` is chosen, then the election officials will first `scan votes`, then `override software`. This scenario occurs when a batch of ballots is scanned more than once by mistake, and, to correct the erroneous vote tally calculated, the election officials must manually override the software. Although the model makes a reference to a manual override of software, no technology assumptions are made here. Note that `scan votes` looks differently from other steps, lacking for example the filled circle above its name–this is because `scan votes` is a *reference*, or an invocation of a step that has already been declared elsewhere in the process, in this case as a child of the `perform ballot count` step. References behave in exactly the same way as the original declarations of a step, and require the same number and types of input and output parameters, and an agent of the same type.

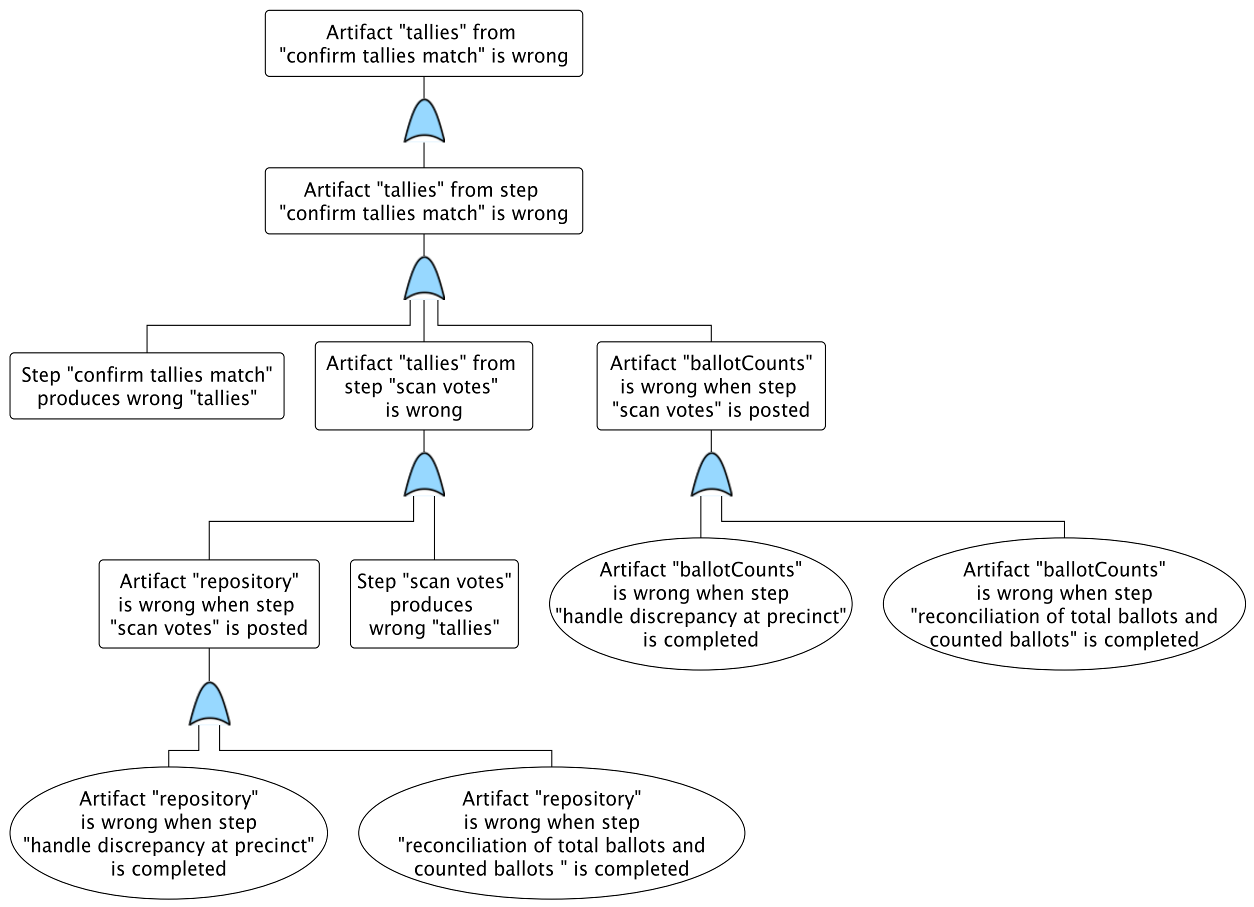
If the election officials decide to `perform random audit` instead of `rescan`, then the `Vote Count Inconsistent Exception` is handled differently. Note that `perform random audit`, like `scan votes`, is a reference and is originally declared as a child of the `perform ballot count` step. Note that in the context of the exception handler, there is no question mark on the edge leading to the reference of `perform random audit`, indicating that it is not optional when instantiated in this context. Also, recall that the original declaration of `perform random audit` can throw a `Vote Count Inconsistent Exception`, and therefore the reference can do the same. If this happens, the exception is propagated up the call stack, as in a traditional programming language, until an appropriate handler is encountered. In this case, the exception would be handled by the `do recount` handler shown in Figure 1, indicating that the discrepancy could not be resolved at the precinct level and must be escalated.

3.4 Fault Trees

As noted above, continuous process improvement relies upon the identification of defects, which is then followed by attempts to remove the defects, and subsequent verification that the defects have indeed been removed. We now present one form of analysis that can be used to trigger such an improvement, namely the analysis of the impact of incorrect performance of a process step. The effects of incorrect step performance can be studied by creating fault trees. These structures can be automatically generated from a Little-JIL process definition, and the derived fault tree can then be used to determine all the various combinations of incorrectly performed steps that can lead to the occurrence of a specified hazard. A "hazard" is a condition under which it becomes possible for an accident that causes substantial damage or loss of life to occur. In this paper we study how various hazards can occur, given that the model being analyzed is a correct representation of an actual real-world process.

Figure 3 depicts the highest levels of a fault tree generated from the process defined in Figure 1, for the hazard "artifact `tallies` produced by step `confirm tallies match` is wrong on output." Only a small part of the fault tree is presented here for simplicity and readability, but the full fault tree specification is made available online. The full tree can be automatically derived from the full process specification, of which similarly only a part has been presented here. Two kinds of nodes comprise a fault tree: event nodes represented as rounded rectangles or ellipses, and decision nodes, represented as gates using standard notation. An AND gate means that in order for its parent to occur, all of its children must occur; an OR gate means that only one of its children needs to occur for the OR gate's parent to occur; and a NOT gate indicates the negation relationship. Events depicted by rounded rectangles indicate leaf events in the full tree; ellipses indicate events that have further decomposition, not included here.

The fault tree in Figure 3 indicates only a few possible scenarios that would result in the wrong tallies being reported after the step `confirm tallies match` completes. For example, following one scenario in the tree from the bottom up, if the ballot repository is somehow corrupt when the election officials complete the `reconciliation of total ballots and counted ballots`, then the ballot repository passed on to the next step, `scan votes`, will be incorrect. In turn, this will re-



Artifact "tallies" from "confirm tallies match" is wrong
Artifact "tallies" from step "confirm tallies match" is wrong
Step "confirm tallies match" produces wrong "tallies"
Artifact "tallies" from step "scan votes" is wrong
Artifact "ballotCounts" is wrong when step "scan votes" is posted
Artifact "repository" is wrong when step "scan votes" is posted
Step "scan votes" produces wrong "tallies"
Artifact "ballotCounts" is wrong when step "handle discrepancy at precinct" is completed
Artifact "ballotCounts" is wrong when step "reconciliation of total ballots and counted ballots" is completed
Artifact "repository" is wrong when step "handle discrepancy at precinct" is completed
Artifact "repository" is wrong when step "reconciliation of total ballots and counted ballots " is completed

Figure 3: Fault tree for the hazard “artifact `tallies` produced by step `confirm tallies match` is wrong on output.”

MCS	Events in the MCS
MCS A	Step `scan votes` produces wrong `tallies`
MCS B	Step `confirm tallies match` produces wrong `tallies`

Table 1: Smallest MCSs calculated for the fault tree in Figure 3; two SPFs are identified.

sult in wrong vote tallies reported by `scan votes`, and the wrong tallies will then be propagated to `confirm tallies match`, which will report the wrong tallies upon completion, resulting in the hazard.

3.5 Automatic Generation of Fault Trees and Minimal Cut Sets (MCS)

Given that the full fault tree generated for this hazard contains over seventy nodes, the ability to automatically derive these structures from Little-JIL process definitions is very valuable. Once a fault tree is derived, it can be manually inspected to identify different vulnerability scenarios that could lead to the hazard occurring. Perhaps more important, however, fault trees are also amenable to an automated technique for calculating Minimal Cut Sets. A Cut Set is a combination of events such that if all events in the cut set occur, the hazard will occur. A Minimal Cut Set (MCS) is a Cut Set for which the removal of any event causes the set to no longer be a Cut Set. Thus, a MCS of size two identifies a combination of two events that together will lead to the occurrence of the hazard. A MCS of size one therefore indicates a single point of failure (SPF). SPFs would seem to be particularly worrisome sources of process vulnerability because they indicate single steps in the process that, when executed incorrectly, can lead to the hazard.

MCSs can be readily and automatically calculated from the fault tree as follows. Starting from the root node, for each gate in the fault tree a Boolean equation is constructed, with the parent node on the left side and the child nodes on the right side. If the gate is an OR, the children nodes are connected by the Boolean operator +, or if the gate is an AND, children nodes are connected by the Boolean operator $*$. NOT gates only negate single leaf nodes, so the negated versions of those nodes are substituted in appropriately. These equations are then computed using standard Boolean algebra, where 1 is equivalent to $true$, and 0 is equivalent to $false$, and a NOT operator converts a 0 to 1 ($\neg false = true$) or a 1 to 0 ($\neg true = false$). so that if a clause is made up of terms connected by the $*$ operator, all its terms would have to be 1s for the clause to be $true$, and if the terms are connected by a +, at least one of the terms in the clause must be a 1 for the clause to evaluate to $true$ (for example 1+1+0 = 1, and 0$*$1$*$1 = 0).

If a node on the right side of the equation is further decomposed in the fault tree, then the node is replaced by its corresponding elaboration equation as described above. This recursive substitution of elaborating equations as prescribed by the fault tree structure is repeated until all possible substitutions have been made, resulting in a Boolean equation consisting of only the hazard (i.e. root node in the fault tree) on the left side of the equation, and only simple events with no further decomposition (i.e. leaf nodes in the fault tree) connected by + and $*$. The resulting right side expression is then transformed into a more compact representation using standard techniques for Boolean expression minimization such as the ones described in [42] and finally transformed into disjunctive normal form (a disjunction of conjunctive clauses). Given this equation, then the hazard will occur (i.e. evaluate to 1 or $true$) only if one or more of the conjunctive clauses evaluate to $true$, which can only happen if all the terms in a conjunctive clause evaluate to $true$, indicating all participating simple events in that clause occurring. Therefore each conjunctive clause forms a MCS.

3.6 The Improvement Loop

The technologies just described support the precise modeling of election processes and the analysis of such models for the presence of certain types of problematic vulnerabilities. This framework, when applied iteratively, can provide suitable support for continuous process improvement. Once problems have been identified through the application of different analyses, process modifications can be suggested to address these problems. These modifications are usually identified by the domain experts through discussions of the model, to ensure that the proposed modifications are reasonable, would not interfere heavily with the real-world process, and would be easy to effect. The proposed process modifications can be evaluated before being deployed in the real world, by making the appropriate changes to the process model, and then reanalyzing the modified model to ensure that the changes successfully correct the problems, without introducing more problematic vulnerabilities in other parts of the process. Because the process is modeled using a precise and rigorous process definition language, multiple kinds of analysis can be applied to discover different kinds of problems, and the fault-tree analysis presented in this paper is only one example.

In the case where a SPF is discovered in the process model, remedial changes often add redundancy to the process by means of additional checks and balances. The goal of such changes is to increase the size of the MCS, or the number of events that must occur together to cause the hazard to occur. Choosing where to place such additional checks and balances often requires the input of domain experts as well, to ensure that redundancy is added judiciously in a way that is unobtrusive but still prevents the SPF. This is usually done by identifying several candidate steps that are SPFs or participate in small MCSs and then consulting with election officials to identify steps that are often performed incorrectly in the real-world process and would therefore benefit the most from extra redundancy, versus steps that have, in the election officials' experience, a lower chance of being carried out erroneously, or that are performed so infrequently that added redundancy would not have the same impact than it would at steps that are performed very frequently.

4 Case Study

To demonstrate our approach, we have selected one problem that Yolo County has encountered, namely vote tally discrepancies during the counting of votes. We use our model to show how the problem can arise, and that the probability of this problem occurring is increased by the presence of single points of failure within the process model. We indicate how to implement corrective procedures, and then analyze these procedures to ensure that the corrected model no longer exhibits single points of failure.

4.1 A Potential Problem in a Real-World Election Process

The potential problem arises in the counting of ballots, and was identified by elections officials at Yolo County as a source of discrepancies. Yolo County counts all ballots at Election Central. A brief description of the real-world process follows.

At the beginning of the day, the election officials record the number of blank ballots sent to each precinct. During the election, every voter has to sign the voting roll that contains the list of registered voters. The voter is then given a ballot, goes to a voting booth, votes, and then puts the ballot into the ballot box. If a voters mis-marks a ballot, a new ballot will be issued upon request (up to three ballots). The mis-marked, or spoiled, ballots are kept separately from the other ballots. If the voter chooses to vote on an electronic voting machine (DRE+VVPAT), no paper ballot is used; instead, the DRE+VVPAT records the ballot both electronically and on a paper tape.

If a voter is not listed in the voting roll, or there is a question about whether the voter should be allowed to vote at that precinct, the voter can cast a provisional ballot. In Yolo County, provisional ballots cannot be cast electronically; they must be cast on paper.

When the polling station closes, the poll workers assemble the ballots into four groups: cast, spoiled, provisional, and unused. The number of ballots in each group is recorded, and the total is checked against the number of ballots given to the precinct. The ballots are then taken to Election Central, where the counts are recomputed and rechecked.

At this point, the election officials place the cast ballots into clear bags, one for each precinct. The bags are moved to a room with high-speed scanners. Each bag is individually opened and the ballots scanned. Scanning software counts the votes cast in each race, producing vote totals. Both the number of ballots scanned and the totals for the races are recorded separately from the initial ballot count reported by the precinct. Note that this is a description of the real-world process and as such it describes the specific voting technologies used. The process model, however, is technology-agnostic; that is, although it specifically models a central-count optical scan (CCOS) process, it makes no assumptions about the devices being used, so none of the following results are dependent on the specific scanner devices or scanning and tabulation software that Yolo County uses.

We consider the following problem:

- *Discrepancy between precinct and Election Central counts*. This problem occurs when the number of ballots counted at and reported by the precinct or polling station is inconsistent with the number of ballots or vote tallies reported by the scanner at Election Central.

4.2 How the Problem Can Happen in the Real-World Process

A number of occurrences may cause the problem. The two most likely ones are that ballots are misplaced while they are being transported from the precinct to Election Central, or that the identifying information about the precinct from which the ballots came is incorrect. The latter will cause the ballots to be counted as though they came from precinct X when they should be counted as having come from precinct Y, throwing off the counts for both precincts. Other possible problems are that the ballots could be miscounted, either at the precinct or at Election Central. Even if automated, the systems that do the counting may err due to faulty use, mechanical problems, or other reasons. If any of the counts are done by a computerized system, faulty software may misread the vote tallies from the storage devices. More simply, the handwriting of the ballot counts on the precinct summary sheets may be hard to read, or be misread. A more sinister reason is that someone adds ballots, or removes ballots, from the set of ballots to be counted.

4.3 Applying Fault Tree Analysis to Identify How the Problem Can Happen in the Model

The fault tree generated for the hazard "artifact `tallies` produced out of step `confirm tallies match` is wrong on output" is shown in Figure 3. Note that the first few levels of this fault tree contain only OR gates. This is typically undesirable because it indicates that only one of possibly several child events has to occur in order for the parent event to occur, and this can lead to a SPF. In fact, Table 1 lists two SPFs identified among the MCSs calculated for the fault tree in Figure 3. Indeed, the complete list of MCSs (omitted here due to space limitations but made available online) contains sixteen Cut Sets, twelve of which are of size two or smaller.

Note that both SPFs refer to the incorrect execution of steps in the `count votes` subprocess shown in Figure 2. One way that the hazard may occur is if the step `confirm tallies match` itself is carried out incorrectly. This is a SPF as there is no mechanism in the process to detect when this step has been performed incorrectly, either through error, or as the consequence of collusive election official behavior. Domain experts must identify the different ways that a step may be executed incorrectly and would presumably use that information to suggest effective ways to change the process to remove the SPF.

Another way for the hazard to occur is if the step `scan votes` produces the wrong vote tallies. An example of how this may occur would be if a batch of ballots was erroneously scanned twice. But there are many other ways for this step to produce wrong tallies. For example the error might be caused by a faulty scanner, or by tampering with the ballot repository. The full fault tree includes these scenarios identified by election officials, as well as others.

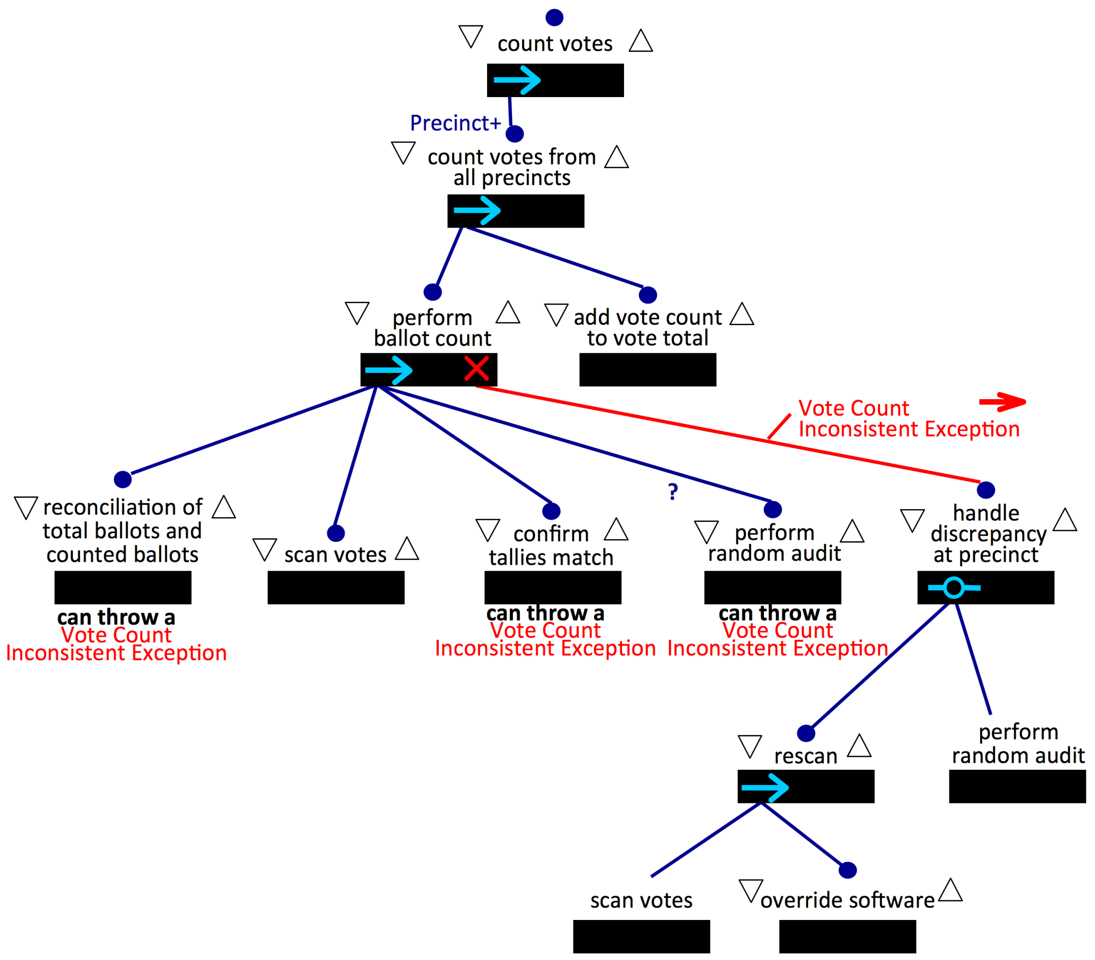

count votes
Precinct+
count votes from all precincts
perform ballot count
add vote count to vote total
Vote Count Inconsistent Exception
reconciliation of total ballots and counted ballots
can throw a Vote Count Inconsistent Exception
scan votes
confirm tallies match
can throw a Vote Count Inconsistent Exception
?
perform random audit
can throw a Vote Count Inconsistent Exception
handle discrepancy at precinct
rescan
perform random audit
scan votes
override software

Figure 4: The `count votes` subprocess, augmented with an additional exception declaration at the step `confirm tallies match`.

4.4 Process Model Modifications to Support Continuous Process Improvement

Both SPFs identified in Table 1, for the fault tree in Figure 3, result in wrong tallies being reported after the step `confirm tallies match` completes execution. This incorrect result is reached in one of two ways: either the step `confirm tallies match` itself is carried out incorrectly or it receives the wrong tallies from its predecessor, `scan votes` and propagates the mistake. SPFs of this nature can often be removed by the addition of redundancies, or extra checks, to the process. Referring back to the subprocess for `count votes` in Figure 2, note that the steps `reconciliation of total ballots and counted ballots` and `perform random audit` both have the ability to raise a `Vote Count Inconsistent Exception` if a discrepancy arises. The step `confirm tallies match` could also be allowed to raise this exception, as shown in Figure 4, to better handle the situation where there is a mismatch between the tallies reported by scanning the votes and the ballot counts indicated on the precinct summary sheet. This seemingly small change in the process definition leads to a very different fault tree. Figure 5 shows the top few levels of the fault tree derived from the modified process definition, using the same hazard, "artifact `tallies` produced by step `confirm tallies match` is wrong on output." Note that unlike the original fault tree in Figure 3, the fault tree derived from the modified process has an AND gate connecting the event "Artifact `tallies` is wrong when step `confirm tallies match` is completed" to its children, indicating that now both events must occur in order to cause the hazard. In fact, when the MCSs are computed for this modified fault tree, there are no longer any SPFs. As in the previous case, the full list of MCSs contains sixteen cut sets; only two of them, however, are now of size two (shown in Table 2; the primes indicate that these MCSs were calculated for the fault tree derived from the modified process), and there are no SPFs.

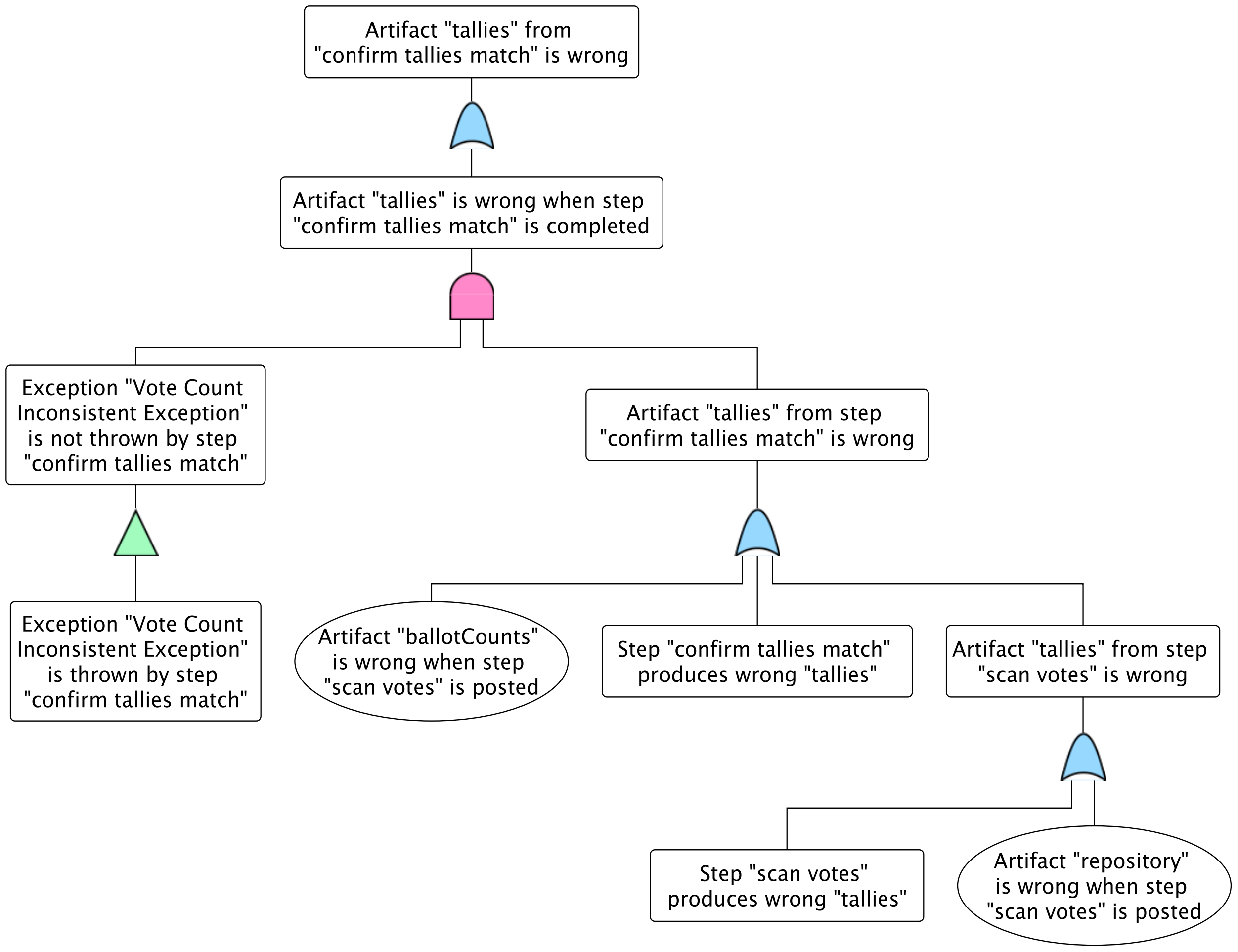

Artifact "tallies" from "confirm tallies match" is wrong
Artifact "tallies" is wrong when step "confirm tallies match" is completed
Exception "Vote Count Inconsistent Exception" is not thrown by step "confirm tallies match"
Artifact "tallies" from step "confirm tallies match" is wrong
Exception "Vote Count Inconsistent Exception" is thrown by step "confirm tallies match"
Artifact "ballotCounts" is wrong when step "scan votes" is posted
Step "confirm tallies match" produces wrong "tallies"
Artifact "tallies" from step "scan votes" is wrong
Step "scan votes" produces wrong "tallies"
Artifact "repository" is wrong when step "scan votes" is posted

Figure 5: After implementing process changes, the fault tree is different for the hazard "artifact `tallies` produced by step `confirm tallies match` is wrong on output."

MCS	**Events in the MCS, separated by semicolon**
MCS A′	Step `scan votes` produces wrong `tallies`; `Vote Count Inconsistent Exception` is NOT thrown by step `confirm tallies match`
MCS B′	Step `confirm tallies match` produces wrong `tallies`; `Vote Count Inconsistent Exception` is NOT thrown by step `confirm tallies match`

Table 2: Smallest MCSs calculated for the fault tree in Figure 5; no SPFs are identified.

It may appear that the SPF would be recognizable in the original process model through manual inspection, and it may seem easy to identify an appropriate process change to remove the SPF. However, note that this example is a simplified case looking at only a small part of the process model, and is only meant to illustrate the approach. Generally, the process model under consideration may be much larger or may include sophisticated parallelism and other characteristics that would make it prohibitively complex for manual inspection to identify every possible scenario that may cause a hazard to occur. In such cases, automatically derived fault trees can provide much needed guidance and identify SPFs that would be very difficult or impossible to spot through manual inspection.

Adding only one additional check eliminates both SPFs identified in the original process model because this extra check is judiciously placed at the step where it would have the most impact. In this case, the wrong tallies may be produced by `confirm tallies match` either because the step itself is performed incorrectly or because it received incorrect tallies from its predecessor, so by placing an extra check at this step, both scenarios would benefit from the redundancy, eliminating both SPFs. Note that an additional check could have been added to `scan votes` instead, preventing it from passing on the wrong tallies to its successor, `confirm tallies match`. Although this would have fixed one SPF, it would not have prevented `confirm tallies match` from producing wrong tallies through its own incorrect performance. A careful consideration of the fault tree and its corresponding MCSs in tandem with the domain experts is needed to identify minimal changes that the election officials feel would be easy to implement within the real-world process that would ensure the removal of a SPF and be most beneficial in preventing further vulnerabilities. These changes can then be made in the process model and evaluated using the same analysis techniques to ensure that they are indeed effective.

5 Conclusion and Future Work

In this paper, we have presented an approach for continuous election process improvement. The approach entails precisely defining a model of the election process that is to be evaluated, and then subjecting this model to different analyses. As a case study, we present the effectiveness of one form of analysis, namely Fault Tree Analysis, for improving the robustness of the election process used in Yolo County, California. Although the case study in this paper concentrates on a model of a counting process that is specific to the Yolo County process, the rest of the model is fairly general and the technologies described in this paper are broadly applicable and are not restricted to a specific process model. Through the application of FTA, we automatically generate fault trees, indicating different scenarios in the process that could enable an undesirable hazard to occur. We then use the fault trees to compute Minimal Cut Sets, combinations of events that could lead to the hazard. Together with election officials, for this case study, we carefully consider SPFs in conjunction with the fault trees and the process model, and identify process changes that would remove the SPFs. To ensure that the modifications successfully remove the SPFs and do not introduce additional vulnerabilities, we reapply the fault tree analysis to the modified process and demonstrate that all SPFs with regard to the specified hazard have been effectively removed by means of a relatively modest process change carefully placed in such a way as to cause relatively large impact.

Continuous process improvement through the application of FTA seems to be an effective approach for improving the robustness of election processes. Careful and precise definition of the election process enables the application of FTA, as well as a plethora of other analysis approaches. One major benefit of using FTA to analyze a carefully defined process model is the automatic generation of fault trees and corresponding MCSs. This is especially important as processes in the election domain are often very large and complex, and manual generation of fault trees would be a onerous and error-prone task. Another benefit is that one process model can be used to consider a number of hazards. Moreover if the process definition is modified it is relatively easy to generate fault trees for the new models. In addition to facilitating FTA, another benefit of a precise process model is that once it is defined, the same model could be used for many different kinds of reasoning, thus amortizing the cost of initial development.

In the future, we plan to explore other ways in which to utilize the process model to identify potential improvements in the real-world process. For example, FTA focuses on identifying different scenarios in the process that could lead to a pre-defined hazard occurring, but there is a complementary technique called FMEA that we intend to pursue. FMEA, or Failure Mode and Effects Analysis, determines what kinds of hazards could occur as a result of the incorrect performance of a pre-defined step (i.e., the failure). There are a number of FTA and FMEA analyses that we intend to pursue, including approaches that combine these two techniques (e.g., [28]). We also plan to model and analyze different jurisdictions' election processes to further evaluate and improve the approach presented in this paper, starting with the election process used in Marin County, California.

Additionally, although not explored in detail in this paper, a carefully defined process model can be used to derive requirements that different agents in the process must satisfy for the overall process to adhere to predefined security or privacy constraints. One obvious benefit of such analysis would be the derivation of requirements that voting technologies and devices would need to satisfy, in order to make guarantees that the devices can be used within the process without leading to undesirable violations of security and privacy constraints. Such a global view is clearly desirable, because it can lead to assurances about the voting devices within the context of the larger election process, in which they play only a small part.

Finally, a jurisdiction might use the approach described here to understand that idiosyncrasies in the way it conducts elections might cause a crucial election process step that is performed by a particular voting device to be a single point of failure. In such a case, the jurisdiction might decide that its voting process should be changed to add redundancy that removes the SPF. We suggest that a core election process that is compatible with elections across many jurisdictions could enable election officials to share information on problems that arise with the use of a specific type of voting technology within that core process–focusing on how the device interfaces with the core election process and without having to discuss intricacies of their own election process or information proprietary to voting device vendors. Such a library of election processes, perhaps accompanied by the results of various analyses, could be made widely available for discussion of best practices for using technologies or procedures to identify and mitigate various problematic vulnerabilities.

Acknowledgements: Thanks to Tom Stanionis, Chief of Staff of the Yolo County Elections Office in California, for his helpful comments and for providing the tabulation process details; to Bin Chen of the University of Massachusetts Amherst, who is the creator of the FTA toolset for Little-JIL, for technical support; to Joshua Molloy of the University of Massachusetts Amherst for modifications to the election process model; and to the EVT/WOTE 2010 reviewers for their helpful comments and suggestions.

This material is based upon work supported by the National Science Foundation under Awards CCF-0905530, CCF-0820198, CNS-0831002, and IIS-0705772. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludäscher, and Steve Mock. Kepler: An extensible system for design and execution of scientific workflows. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM04)*, page 423, 2004.

[2] Tigran Antonyan, Seda Davtyan, Sotirios Kentros, Aggelos Kiayias, Laurent Michel, Nicolas Nicolaou, Alexander Russell, Alexander A. Shvartsman. State-Wide Elections, Optical Scan Voting Systems, and the Pursuit of Integrity. *IEEE Transactions on Information Forensics and Security*, 4(4):597–610, 2009

[3] Earl Barr, Matt Bishop, and Mark Gondree. Fixing federal e-voting standards. *Communications of the ACM*, 50(3):19–24, 2007.

[4] Matt Bishop. Overview of red team reports. Top to bottom review of electronic voting machines, Office of the Secretary of State of California, Sacramento, CA, 2007.

[5] Matt Bishop, Sean Peisert, Candice Hoke, Mark Graff, and David Jefferson. E-Voting and Forensics: Prying Open the Black Box. In *Proceedings of the 2009 Electronic Voting Technology Workshop/Workshop on Trustworthy Computing (EVT/WOTE '09)*, Montreal, Canada, 2009.

[6] Brennan Center Task Force on Voting System Security. *The Machinery of Democracy: Protecting Elections in an Electronic World*. Brennan Center for Justice, New York, NY, 2006.

[7] Phillip J. Brooke and Richard F. Paige. Fault trees for security system design and analysis. *Computers & Security*, 22(3):256–264, 2003.

[8] Jennifer L. Brunner. *Project EVEREST: Evaluation and Validation of Election-Related Equipment, Standards, and Testing*. Office of the Ohio Secretary of State, Columbus, OH, 2007.

[9] Aaron G. Cass, Barbara Staudt Lerner, Eric K. McCall, Leon J. Osterweil, Stanley M. Sutton Jr, and Alexander Wise. Little-JIL/Juliette: A process definition language and interpreter. In *Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland*, pages 754–757, 2000.

[10] Lori A. Clarke, George A. Avrunin, and Leon J. Osterweil. Using software engineering technology to improve the quality of medical processes. In *Companion of the 30th International Conference on Software Engineering*, pages 889–898, 2008.

[11] Compuware Corporation, Columbus, OH. *Direct Recording Electronic (DRE) Technical Security Assessment Report*, 2003.

[12] Bill Curtis, Marc I. Kellner, and Jim Over. Process modeling. *Communications of the ACM*, 35(9):75–90, 1992.

[13] W. Edwards Deming. *Out of the Crisis*. MIT Press, Cambridge, MA, 1982.

[14] Election Assistance Commission, Washington, DC. *2005 Voluntary Voting Systems Guidelines*, 2005.

[15] Election Assistance Commission, Washington, DC. *Election Management Guidelines*, Accessed 2010.

[16] Aaron M. Ellison, Leon J. Osterweil, Lori Clarke, Julian L. Hadley, Alexander Wise, Emery Boose, David R. Foster, Allen Hanson, David Jensen, Paul Kuzeja, Edward Riseman, Howard Schultz, and

Paula Kuzeja. Analytic webs support the synthesis of ecological data sets. *Ecology*, 87(6):1345–1358, 2006.

[17] Clifton A. Ericson. Fault tree analysis—a history. In *Proceedings of the 17th International System Safety Conference*, pages 1–9, 1999.

[18] Federal Election Commission, Washington, DC. *Performance and Test Standards for Punchcards, Marksense, and Direct Recording Electronic Voting Systems*, 1990.

[19] Federal Election Commission, Washington, DC. *Voting Systems Standards*, 2002.

[20] Diimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.

[21] Joseph Lorenzo Hall. Improving the Security, Transparency and Efficiency of California's 1% Manual Tally Procedures. In *Proceedings of the 2008 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT'08)*, pages 1–12, 2008.

[22] Joseph Lorenzo Hall, Luke W. Miratrix, Philip B. Stark, Melvin Briones, Elaine Ginnold, Freddie Oakley, Martin Peaden, Gail Pellerin, Tom Stanionis and Tricia Webber. Implementing Risk-Limiting Post-Election Audits in California. In *Proceedings of the 2009 USENIX/ACCURATE/IAVoSS Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE'09)*, pages 1–24, 2009.

[23] Guy Helmer, Johnny Wong, Mark Slagell, Vasant Honavar, Les Miller, and Robyn Lutz. A software fault tree approach to requirements analysis of an intrusion detection system. *Requirements Engineering*, 7(4):207–220, 2002.

[24] Elizabeth A. Henneman, George S. Avrunin, Lori A. Clarke, Leon J. Osterweil, Chester Andrzejewski, Jr., Karen Merrigan, Rachel Cobleigh, Kimberly Frederick, Ethan Katz-Bassett, and Philip L. Henneman. Increasing patient safety and efficiency in transfusion therapy using formal process definitions. *Transfusion Medicine Reviews*, 21(1):49–57, 2007.

[25] Aggelos Kiayias, Laurent Michel, Alex Russell, and Alexander Shvartsman. *Integrity Vulnerabilties in the Diebold TSX Voting Terminal*. UConn Voting Technology Research (VoTeR) Center, University of Connecticut, Storrs, CT, 2007.

[26] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an electronic voting system. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, pages 27–40, 2004.

[27] Costas Lambrinoudakis, Vassilis Tsoumas, Maria Karyda, and Spyros Ikonomopoulos. Secure electronic voting: The current landscape. In *Secure Electronic Voting*, volume 7 of *Advances in Information Security*, chapter 7, pages 101–122. Kluwer Academic Publishers, Boston, MA, 2003.

[28] Robyn R. Lutz. Software engineering for safety: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, pages 213–226, 2000.

[29] John P. McDermott. Attack net penetration testing. In *Proceedings of the 2000 Workshop on New Security Paradigms (NSPW)*, pages 15–21, 2001.

[30] Rebecca T. Mercuri and Peter G. Neumann. Verification for electronic balloting systems. In *Secure Electronic Voting*, volume 7 of *Advances in Information Security*, chapter 3, pages 31–42. Kluwer Academic Publishers, Boston, MA, 2003.

[31] Lilian Mitrou, Dimitris Gritzalis, Sokratis Katsikas, and Gerald Quirchmayr. Electronic voting: Constitutional and legal requirements, and their technical implications.0 In *Secure Electronic Voting*, volume 7 of *Advances in Information Security*, chapter 4, pages 43–60. Kluwer Academic Publishers, Boston, MA, 2003.

[32] Igor Nai Fovino, Marcelo Masera, and Alessio De Cian. Integrating cyber attacks within fault trees. *Reliability Engineering and System Safety*, 94(9):1394–1402, 2009.

[33] Office of the California Secretary of State, Sacramento, CA. *Top to Bottom Review of Electronic Voting Machines*, 2007.

[34] Sean Philip Peisert. *A Model of Forensic Analysis Using Goal-Oriented Logging*. PhD thesis, Department of Computer Science and Engineering, University of California, San Diego, 2007.

[35] Sean Peisert, Matt Bishop, Sidney Karin, and Keith Marzullo. Toward Models for Forensic Analysis. In *Proceedings of the Second International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE)*, pages 3–15, Seattle, WA, 2007.

[36] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 Workshop on New Security Paradigms (NSPW)*, pages 71–79, 1999.

[37] Nayot Poolsapassit and Indrajit Ray. Investigating computer attacks using attack trees. In *Advances in Digital Forensics III*, volume 242 of *IFIP International Federation for Information Processing*, pages 331–343. Springer, Boston, MA, 2007.

[38] Elliot Proebstel, Sean Riddle, Francis Hsu, Justin Cummins, Freddie Oakley, Tom Stanionis, and Matt Bishop. An analysis of the Hart Intercivic DAU eSlate. In *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, page 3, 2007.

[39] RABA Innovative Solution Cell (RiSC). *Trusted Agent Report Diebold AccuVote-TS Voting System*. RABA Technologies, Columbia, MD, 2004.

[40] Mohammad S. Raunak, Bin Chen, Amr Elssamadisy, Lori A. Clarke, and Leon J. Osterweil. Definition and analysis of election processes. In *Software Process Change*, volume 3966 of *Lecture Notes in Computer Science*, pages 178–185. Springer, Berlin, 2006.

[41] Indrajit Ray and Nayot Poolsapassit. Using attack trees to identify malicious attacks from authorized insiders. In *Computer Security – ESORICS 2005*, volume 3679 of *Lecture Notes in Computer Science*, pages 231–246. Springer, Berlin, 2005.

[42] Richard L. Rudell and Alberto Sangiovanni-Vincentelli. Multiple-valued minimization for PLA optimization. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 6(5):727 – 750, 1987.

[43] Ali M. Rushdi and Omar M. Ba-rukab. Fault-tree modelling of computer system security. *International Journal of Computer Mathematics*, 82(7):805–819, 2005.

[44] Roy G. Saltman. Public confidence and auditability in voting systems. In *Secure Electronic Voting*, volume 7 of *Advances in Information Security*, chapter 8, pages 31–42. Kluwer Academic Publishers, Boston, MA, 2003.

[45] Bruce Schneier. Modeling security threats. *Dr. Dobb's Journal*, 22(12):4–6, 1999.

[46] Walter A. Shewhart. *Economic Control of Quality of Manufactured Product*. D. Van Nostrand Company, New York, NY, 1931.

[47] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 273–284, 2002.

[48] Borislava I. Simidchieva, Matthew S. Marzilli, Lori A. Clarke, and Leon J. Osterweil. Specifying and verifying requirements for election processes. In *Proceedings of the International Conference on Digital Government Research*, pages 63–72. Digital Government Society of North America, 2008.

[49] John A. Simpson and Edmund S. C. Weiner, editors. *The Oxford English Dictionary*. Clarendon Press, Oxford, UK, 2nd edition, 1991.

[50] William E. Vesely, Francine F. Goldberg, Norman H. Roberts, and David F. Haasl. *Fault Tree Handbook*. Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, Washington, DC, 1981.

[51] Oliver Wiegert. *Business Process Modeling and Workflow Definition with UML*. SAP AG, 1998.

[52] Alec Yasinsac, David Wagner, Matt Bishop, Ted Baker, Breno de Medeiros, Gary Tyson, Michael Shamos, and Mike Burmester. *Software Review and Security Analysis of the ES&S iVotronic 8.0.1.2 Voting Machine Firmware*. Security and Assurance in Information Technology Laboratory, Florida State University, Tallahassee, FL, 2007.

[53] Ka-Ping Yee. Building Reliable Voting Machine Software. Ph.D. Dissertation, Technical Report EECS-2007-167, EECS Department, University of California, Berkeley, 2007.

[54] Tao Zhang, Mingzeng Hu, Xiaochun Yun, and Yongzheng Zhang. Computer vulnerability evaluation using fault tree analysis. In *Information Security Practice and Experience*, volume 3439 of *Lecture Notes in Computer Science*, pages 302–313. Springer, Berlin, 2005.