SF Open Source Voting TAC

Official site of the San Francisco Open Source Voting System Technical Advisory Committee (OSVTAC)

Open Source Voting System Project Recommendations

(Approved by OSVTAC on November 16, 2017.)

Last update: December 11, 2017

- Introduction & Table of Contents (for multi-page version)
- Single-page version (long, can be used for printing)

4. Recommendations

4.1. Interim Voting System

- The contract for the interim system (i.e. the system to be used after 2018) should permit all possible combinations of phasing in an open-source system alongside it. Examples of possible combinations include:
  - using open-source components to scan vote-by-mail ballots and the interim system to scan precinct ballots, or vice versa;

    - using an open-source accessible voting device in conjunction with the interim system's precinct-based scanner, or vice versa;
    - scanning the ballots of the interim system using an open-source scanner;
    - tabulating ballots scanned by an open-source scanner using the interim system's tabulation software;
    - using an open-source reporting and/or tabulation system with the output from the interim system's scanners;
    - using open-source components alongside the interim system in some subset of precincts (e.g. for a pilot rollout); or
    - using open-source components alongside the interim system in all precincts (e.g. for an incremental roll-out of the open source system).
- The requirements for the interim system should include interoperability with other systems, and the interoperability formats should be documented so they don't need to be reverse-engineered.

4.2. Incremental Approach

To reduce project risk, complexity, and initial costs, it is important to have a strategy to break the open source voting system project up into smaller, independent deliverables that can be developed and used in real elections before the full system is completed.

This is part of an agile approach and has several advantages. It would let the City start getting real value from the project earlier. It would let the City get confirmation earlier that the project is "on the right track" without necessarily having to commit funds for the entire project. It also builds in a way for the City to take corrective action (e.g. if a vendor developing a particular component isn't performing to expectation). Finally, it eliminates the need to come up with accurate cost and time estimates for the entire project before starting work.

For example, instead of committing $8 million up front for a single project to develop a full voting system, the City could instead start out by spending $2 million on three deliverables, say: one for $1.5 million and two for $250,000. Based on the success or progress of the initial projects, the City could decide to move forward with additional sub-projects, or change its approach (even before the three deliverables are completed). In this way, the City limits its financial exposure to risk.

This section recommends some approaches to achieve this. The purpose of this section is not to serve as an actual plan, but rather to provide concrete suggestions for how the Department can proceed incrementally in developing and deploying an open source voting system.

4.2.1. Possible First Components

The Committee suggests the following as components to start work on and deliver first (see the "Voting System" section for brief descriptions of these components):

1. Results Reporter (Software)
2. Vote Totaler (Software)
3. Ballot Picture Interpreter (Software)
4. Central Ballot Scanner (Hardware & Software)

Choosing the above as first components seems to mirror the approach that Los Angeles County is taking in its VSAP project. In particular, Los Angeles County developed and submitted its "Tally System" for certification even before its in-precinct Ballot Marking Device was engineered and manufactured. Los Angeles County's "RFP Phase 1: #17-008" defines its Tally System on page 48 as–

> A system of hardware and software that reads and captures the vote selections on ballots, applies required business rules and adjudications, tabulates the totals of votes, ballots cast, and other metrics, and publishes the results the election. The tally system also supports transparent auditing processes to ensure the accuracy and integrity of the election tally results.

This seems to encompass the functionality of the four components listed above.

Los Angeles County submitted its VSAP Tally Version 1.0 to the California Secretary of State for certification on September 19, 2017. Section 3.3 (pages 25-28) of its Phase 1 RFP provides more detail on the completion of Los Angeles County's Tally System in relation to other components like their Ballot Marking Device.

4.2.2. Rationale

Below are some reasons for selecting the components above:

- Each component has relatively few dependencies.
- The components are on the easier side to implement.
- The components are independently useful and so can help prove the value of open source.
- The components can be worked on in parallel. Their development can also be staggered in conjunction with other deliverables. For example, development on other components can be started before these are finished.

- In each case, there is open-source code that already exists that development of the components might be able to start from, or at least learn from.

- Working on the components will help to work through and resolve core issues that need to be worked out anyways

- Each of these components supports incremental deployment. Each component can be deployed and used by replacing the corresponding component of a non-open-source interim system, and then interoperating with the other components of the voting system (interim or not). This is true even without requiring anything extra of the interim system. See the "Deployment Strategies" sub-section below for further details.

  In contrast, an example of a component that probably *wouldn't* support incremental deployment as easily is the ballot layout software application. This is because an interim system's scanners probably can't be guaranteed to scan ballots created by a third-party.

  Similarly, it is probably more difficult to design an accessible ballot-marking device that can mark another vendor's ballot than it is to design a scanner that can interpret another vendor's ballot. This is because marking a ballot is a harder problem to solve than interpreting a ballot. While the latter is primarily a software problem (which would be addressed by the ballot image interpreter component), the former leans more towards being a hardware problem.

For the Results Reporter:

- The results reporter is probably the "easiest" component to implement and has the least amount of risk, since it is responsible merely for formatting and presenting information. In this way, it would be a good warm-up project.

- Since many members of the public view the Department's election results pages online, it would nevertheless be a highly visible use of open-source software.

- It could also be a good public outreach / educational tool around open source and the open source voting project. The Department could solicit feedback from the public on how the results pages could look or be improved, and the Department could implement the best suggestions (since the reporter would be open source).

- Making the reporter open-source would also be inherently useful because it would give the Department the ability to customize and improve the current format, and accept contributions from the public.

For the Vote Totaler:

- This component is also one of the easiest components and so would be good to start with.

- This is also a component that other jurisdictions would be able to use and benefit from relatively easily (e.g. jurisdictions using RCV would be able to use the RCV algorithm functionality). In this way, other jurisdictions could start to understand the benefits of open source.

For the Ballot Picture Interpreter:

- This is a core software component that would be used in a number of different components, so it is natural to start working on it first.
- Even in the absence of deployed open-source hardware components, it could be used by members of the public to "check" the scanning done by the interim system, provided the digital ballot pictures are made public.
- The open-source software OpenCount might go a long way towards implementing this component.

For the Central Ballot Scanner:

- This is probably the "easiest" hardware component to work on and implement first, for reasons that will be described below.
- Deploying this component alone would result in a majority of votes being counted by open-source software. For example, in the November 8, 2016 election 63% of ballots were vote-by-mail (263,091 out of 414,528 ballots in all). In this sense, this component provides the biggest "bang for the buck."
- This component doesn't require answering the question of whether to use vote centers, since vote-by-mail ballots need to be tabulated centrally whether or not San Francisco moves to a vote-center model.
- Unlike precinct-based hardware components like the accessible voting device and precinct-based scanners, this hardware component would be operated in a more controlled environment with more highly trained staff. As a result, it also doesn't need to meet the same portability, durability, usability, and transportation requirements as precinct-based equipment (which also might require a custom casing or shell in the case of precinct equipment).
- Also unlike precinct-based hardware components, fewer units would need to be purchased or manufactured, so it is probably less costly and expensive to do this step first. For example, for comparison, San Francisco currently has four high-speed central scanners, but around 600 precincts.

- Central scanners provide multiple possibilities for incremental rollout, including using the component alongside and in parallel with the interim system, which all help to mitigate risk. These approaches are described in the "Deployment Strategy" section.
- Implementing the central scanner before the precinct scanner also makes sense from a software dependency perspective. The central scanner includes most of the software that an in-precinct scanner would need anyway, like ballot interpretation, understanding election definition and ballot layouts, etc. However, the central scanner provides a safer and more controlled environment in which to exercise these code paths for the first time. In other words, with the exception of the high-speed and high-volume nature of the hardware, it is a strictly simpler component than the precinct-based scanner.

4.2.3. Component Details

This section lists more details about each of the four components we suggested above. For each of these deliverables, we provide—

- A rough level of the relative complexity (low / medium / high),
- A brief description (though this already appears for the most part in the Background section of this document),
- What components the deliverable interacts with,
- Possible interfaces / data formats that might be needed,
- Sub-components,
- Dependencies from a project management perspective (i.e. what might be needed in advance), and
- Other outcomes / deliverables associated with delivering the component.

4.2.3.1. Results Reporter (Software)

Complexity: Low

Description. This is a software-only component responsible for generating human-readable reports in various formats from structured results data.

Interfaces / data formats. Needs to accept as input:

- the "election definition" data (e.g. contests, candidates, districts, etc.).
- the vote total data for the contests as a whole as well as at the desired aggregation levels (e.g. neighborhood, precinct, district, election day vs. vote-by-mail, etc.), including the round-by-

round vote totals for RCV elections.

Sub-components. The reporter should be able to generate:

- the Statement of Vote (e.g. in PDF format),
- tables for the Election Certification letter (e.g. in PDF format),
- HTML pages for the Department website, and
- Possibly also reports to facilitate the public observation and carrying out of post-Election Day audit processes (e.g. vote totals divided by batch or precinct).

Other outcomes / deliverables. The required input data and formats should be spelled out.

Possible dependencies / pre-requisites. Real data from past elections for proto-typing and testing.

4.2.3.2. Vote Totaler (Software)

Complexity: Low

Description. This is a software-only component responsible for aggregating vote data and generating election results in a machine-readable format. This includes running the RCV algorithm to generate round-by-round results.

Interfaces / data formats. Needs to accept as input:

- the "election definition" data (e.g. contests, candidates, districts, etc.).
- cast vote records (aka CVR's) for all ballots.

Sub-components.

- the code responsible for running the RCV algorithm could be its own component.

Other outcomes / deliverables. The required input data and formats should be spelled out.

Possible dependencies / pre-requisites. Real data from past elections for proto-typing and testing.

4.2.3.3. Ballot Picture Interpreter (Software)

Complexity: Medium

Description. This is a software-only component responsible for interpreting digital ballot pictures, namely by generating a cast vote record (CVR) given a digital picture of a ballot. The component must support ballots from “third-parties” (e.g. the interim voting system) to support incremental roll-outs like pilot and hybrid rollouts. The open-source software OpenCount developed at UC Berkeley could be a foundation for this.

Applicability. This component can possibly be used in the following components:

- precinct ballot scanners
- central ballot scanner
- software application for adjudicating or auditing ballots using their digital pictures, independent of a hardware scanner.

Interfaces / data formats. Needs to accept as input:

- the “election definition” data (e.g. contests, candidates, districts, etc.).
- the “ballot layout” data (e.g. where contests are located on each ballot card for each ballot type, etc.).
- the digital ballot pictures themselves.

Needs to output for each ballot:

- a cast vote record (CVR) of the markings on the ballot.

Sub-components. This component can possibly have the following sub-component:

- a “contest-unaware” interpreter that accepts a digital picture of a ballot and ballot layout data and outputs what markings are on the ballot (e.g. what bubbles are filled in, independent of their contest or candidate meaning).

Other outcomes / deliverables. The required input data and formats should be spelled out.

Possible dependencies / pre-requisites. Real data from past elections for proto-typing and testing.

4.2.3.4. Central Ballot Scanner (Hardware & Software)

Complexity: High

Description. This is a hardware component responsible for high-speed, high-volume ballot scanning in a controlled environment under staff supervision (e.g. vote-by-mail ballots). It should be capable of (1) exporting CVR’s and digital pictures of the ballots it scans, (2) “out-stacking” ballots that require manual inspection or handling, and (3) possibly printing unique identifiers on each ballot when scanning to support the auditing of individual ballots.

Interfaces / data formats.

- Same as for the Ballot Picture Interpreter.
- Also needs to store digital pictures of ballots in a defined image format.

Sub-components.

- Device drivers (software API's to control low-level scanner functionality and, if present, the printer).
- Ballot picture interpreter (see component description above).
- High-level software to orchestrate calls between the device drivers and the ballot picture interpreter.
- Printer component to print unique identifiers (possibly required).

Other outcomes / deliverables. The required input data and formats should be spelled out.

Possible dependencies / pre-requisites. Real data from past elections for proto-typing and testing. Samples of ballots from past elections and/or the interim voting system.

4.2.4. Deployment Strategies

The components listed above can be deployed and used in conjunction with a non-open-source interim system even before a full open-source voting system is ready. This section provides more details about how this could be done.

For example, an open-source results reporter could be used to report the election results of the non-open-source interim system. It would simply need to take in the aggregate, numeric results from the interim system. The output would not need to interact with the interim system.

Similarly, an open-source vote totaler could be used to compute the numeric results of an election run with the interim system. It would only require taking in the non-aggregated numeric results from the interim system, and then feeding the aggregate results into the results reporter.

4.2.4.1. Central Ballot Scanner Phases

For the central ballot scanner, there are a number of options for incrementally phasing in an open-source version.

In chronological order, some of these possible phases are–

1. Even before the scanner hardware is ready to be tested, the software-only ballot image interpreter component could be used to check the vote counts of the interim system from the information of the digital ballot pictures. In addition, if the pictures are made public during the canvass (along with the ballot image interpreter software), even members of the public could perform this "check."

2. When the open-source central scanners are ready enough to test, the scanners could be used to scan vote-by-mail ballots *in addition* to the interim system scanning them. This could be used both to check or audit the interim system, as well as to test the open-source scanners. This can likely be done even without certifying the scanners. This is essentially what the Humboldt County Elections Transparency Project did in the late 2000's.

3. Once we have enough confidence in the open-source scanners, they could be used as the primary scanner for *some* of the vote-by-mail ballots (e.g. in a pilot of the open-source scanners that precedes a full-scale rollout). This option could possibly be done prior to certifying the scanners, by taking advantage of California bill SB 360 (2013-2014).

4. Finally, once the open-source central scanners are certified, they could be used to scan *all* of the vote-by-mail ballots (while the interim system could be responsible for counting in-precinct ballots). In this scenario, the interim system could perhaps even be used as a fail-safe backup in case of an unexpected issue with the open-source system (or else as a check, in the same way that the open-source scanners were used as a check in bullet point (2) above).

4.3. Requirements-gathering

This section contains recommendations related to gathering requirements. For committee recommendations of specific requirements, see the Requirements section below.

4.3.1. Key Decisions

The following are some key decisions about requirements that need to be made at some point when designing and developing the voting system.

4.3.1.1. Vote Centers

California SB 450 ("Elections: vote by mail voting and mail ballot elections") authorizes counties to conduct elections using vote centers. The Department of Elections should develop a sense as soon as possible of the likelihood of using vote centers because that could affect the requirements and design of the system. Making this decision earlier could decrease costs since the design and development wouldn't have to cover multiple scenarios.

4.3.1.2. Pre-printed versus on-demand ballots, including how selections are marked

For in-person voting, the question of pre-printed ballots versus on-demand ballots, combined with how ballots are marked (for both accessible voting and not-necessarily-accessible voting) will greatly affect what type of precinct hardware needs to be developed. It also greatly affects how many units would need to be purchased and deployed per precinct.

This decision needs to be made separately for accessible voting and not-necessarily-accessible voting. However, the decisions for the two scenarios are not independent. They are related.

For not-necessarily-accessible voting, options include—

1. Pre-printed ballots with selections marked by hand
2. On-demand ballots printed without selections and marked by hand
3. On-demand ballots printed together with selections using an accessible device

For accessible voting, options include—

1. Pre-printed ballots marked using an accessible device (e.g. by inserting the ballot)
2. On-demand ballots printed without selections and marked using an accessible device
3. On-demand ballots printed together with selections using an accessible device

Some considerations include—

1. The more that the accessible and not-necessarily-accessible scenarios are similar to one another, the more consistent the voter experience will be. The most similar would be if both scenarios are conducted with option (3), "on-demand ballots printed together with selections using an accessible device." Different but still similar would be if both groups use pre-printed ballots or on-demand ballots printed without selections, with the only difference being how the ballot is marked (by hand versus using an accessible device). The least similar would be, for example, option (1) for not-necessarily-accessible voting and option (3) for accessible voting. The latter happens to be how San Francisco conducts its elections today.

2. To preserve ballot secrecy during the count, it is preferable if the voted ballots "look" the same across the accessible and not-necessarily-accessible methods. An example of the ballots looking different would be if accessible voting results in voted ballots that contain only the voters' selections and not other ballot choices, whereas the not-necessarily-accessible approach results in voted ballots containing all ballot choices but with the voters' selections marked.

3. Requiring ballots to be printed on-demand for all voters (either with or without selections) would require using a printer for every voter in the polling place. This would likely require

more electronic devices at each polling place, which in turn would increase costs, complexity, and the possibility of something breaking or going wrong. These printing requirements would be even greater for the case of printing not just blank ballots for all voters, but ballots with their selections for all voters. This is because voters would likely need to be occupying a machine while they are making their selections.

4. Using pre-printed ballots allows voters without disabilities to vote using the “low-tech” solution of only using a marker or pen (with the exception of the precinct ballot scanner that normally scans and counts the ballot). This would reduce the polling place’s overall dependency on technology and possible things that can go wrong (e.g. power outages, one or more machines breaking, etc.).

5. Using pre-printed ballots results in increased paper usage and printing costs, since the Department needs to prepare extras of every ballot type (including every language, party preference, and combination thereof).

6. Printing ballots on-demand would theoretically allow voters to get the correct ballot type even if they go to the wrong precinct. Currently, a voter going to the wrong precinct can only choose among the ballot types pre-printed and made available at that precinct.

7. If ballots are printed on-demand, poll workers would not have to keep track of all the different ballot types (e.g. different languages, the various party ballots, etc.). It would instead automatically be taken care of by the ballot printer.

8. If the accessible device is a ballot-marking device, the device will be harder to use because each ballot card would need to be inserted individually into the device. Conversely, if the accessible device prints the ballot with selections, fewer physical cards would be required.

4.3.1.3. Printing unique identifiers on ballots at scan-time

One key decision is whether a unique identifier should be printed on every ballot while it is being scanned.

Pros:

- This would permit more sophisticated auditing approaches that involve selecting individual ballots at random, which could reduce time and costs (e.g. risk-limiting audits). Without this feature, auditing needs to be done in larger “batches,” or ballots need to be kept in careful order to allow accessing individual ballots.

Cons:

- It is not clear if COTS scanners support the feature of printing while scanning.

- The scanner hardware would become more complicated since there would be another "moving part" that can break.

4.3.1.4. End–to–end verifiability

It should be determined how much additional work would need to be done to make the voting process end-to-end verifiable, and whether and which designs are more compatible (e.g. among approaches listed in section 5.3.1.1. "pre-printed versus on-demand ballots"). Also, is this something that could be incorporated later on in the process, or does it need to be incorporated from the beginning?

4.4. Requirements

This section lists some of the requirements the system should satisfy.

4.4.1. Accessibility

- In addition to an audio component and touchscreen, the voting system should support accessible features including, but not limited to: sip and puff input, a keyboard for write-in votes, voice activation, synchronized audio and video, joystick input, Tecla switch, and tactile buttons. These two letters from Mr. Fred Nisen (Supervising Attorney for Voting Rights, Disability Rights California) provide more detail.

4.4.2. Other

- [TODO: should we recommend (1) supporting manually marked ballots in the polling place, or (2) requiring the use of a computer ballot-marking and/or ballot-printing device?]

- [TODO: should we recommend (1) pre-printed ballots at polling places, or (2) printing ballots on-demand?]

- [TODO: should we recommend for or against end-to-end verifiability?]

4.5. Project Management

- The Department should hire a staff person to be in charge of managing the project. The person should have experience and expertise in managing technical projects of a similar size and complexity.

- As soon as possible, the Department should develop and publicize a rough project plan and timeline for the development and certification of an open source system, for the case that the project is funded. It is okay for this plan to be tentative. It can be refined over time as more information becomes available. Articulating even a tentative plan would also help in crafting an RFP for the interim system.

- For deliverables, favor smaller deliverables that can be tested independently of other components. In particular, if developing a software application, it may make sense for one or more of the underlying libraries to be delivered separately and/or earlier, rather than the application as a whole being the only software deliverable.

  One example is an application to tabulate the results of an RCV contest. The code responsible for running the algorithm could be delivered and tested as a stand-alone library separate from any user-interface.

  Another example is an application to adjudicate ballots. The code for automatically interpreting the digital ballot picture could be separated out as its own library. Indeed, this corresponds to the Ballot Picture Interpreter software component.

  [TODO: add a comment about the vendor providing a UI shim to support the testing of software libraries.]

- [TODO: think about the division of responsibilities between the City and vendor. For example, who should be responsible for project management—the City or a vendor?]

- [TODO: provide specific recommendations around agile.]

4.6. Open Source

This section covers topics related to open source.

- Each software component being developed should be licensed under an OSI-approved software license, with a copyleft license being preferred (see also the Facts & Assumptions section).

- All software development should occur in public (e.g. on GitHub), rather than, for example, waiting for the software to reach a certain level of completion before becoming public. (See also item (b) of the third “resolved” paragraph of the Commission’s Open Source Voting Resolution.)

- All software being developed in public should have an open source license when development first starts, rather than, for example, adding a license file later on. This would eliminate any confusion and uncertainty from members of the public as to whether the software will really be open source. This would encourage members of the public to start contributing to the project as early as possible.

- All software being developed should be developed using an open source programming language and toolchain. This means an open source compiler or runtime should be available for the language(s) used, and it should be possible to build and run the software from source using only open source tools. For programming languages and build tools, any OSI-approved license should be okay; they need not be copyleft.

- Reuse of existing open source libraries, tools and software is encouraged. Any such pre-existing third-party code used should be available under an OSI-approved license, but need not be copyleft. If modifications to third-party code are developed, and the original third-party code has a different license than the main software's license, the modifications should be dual-licensed under both licenses, if possible. (See also item (e) of the third "resolved" paragraph of the Commission's Open Source Voting Resolution.)

- The aggregate system (including the infrastructure, stack, and services) should be open source. This includes but is not limited to things like the operating system, database, web server, etc, if present.

- In addition to the software being open source, project documentation should be openly licensed. This includes things like design documents, installation and setup documents, user manuals, and testing documents. The recommended license for documentation is the Creative Commons Attribution-ShareAlike 4.0 license (CC-BY-SA 4.0). (See also the reference to "freely and openly licensed" documentation in the Commission's Open Source Voting Resolution.)

- [TODO: provide recommendations related to managing community feedback and contributions during project development. Also think about whether contributor license agreements (CLA's) should be required.]

4.7. Procurement

[TODO]

4.8. Software architecture and design

- When defining software components to develop, favor designs that promote reusing components. For example, a software library that can read a digital ballot picture and return the marked "votes" (what we are calling a "ballot picture interpreter" component) can be used in both precinct scanners and central scanners (as well as software applications for adjudication or auditing). Favoring component reuse can mean having less code to write and test, which in turn can reduce required time and costs.

4.9. Software development

- The project should not depend on volunteers for the successful completion or security of the project. However, useful volunteer contributions should be encouraged and not turned away.

4.10. Hardware design

[TODO]

4.11. Documentation

[TODO]

4.12. Security

[TODO]

4.13. Testing

- Datasets of real election data (e.g. a couple past elections in San Francisco of different types) should be compiled in a structured format for product prototyping and testing. This includes not just vote totals but also candidate and contest data. This will help in establishing requirements and designing the system.

4.14. Certification

[TODO]

4.15. Hardware manufacturing or assembly

[TODO]

4.16. Deployment

[TODO]

4.17. Software maintenance

[TODO]

4.18. Hardware maintenance

- The City should prefer professional, commercial support for maintaining the aggregate system (including the operating system, stack, and software services, etc.) over "in-house" maintenance – even though the components are open source. This will make it easier, for example, to ensure that security patches are applied on a timely basis. An example of such a provider is Red Hat.