September 19, 2017

From: Chris Jerdonek, OSVTAC Chair

To: Open Source Voting System Technical Advisory Committee (OSVTAC)

RE: Agenda Item #9 – Proposed text re: Incremental Approaches

---

5. Recommendations

5.1 Interim Voting System

…

5.2 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 an accurate estimate 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: 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.

5.2.1. 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 Image Interpreter (Software)
4. Central Ballot Scanner (Hardware & Software)

These components can be developed 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.

5.2.1.1. Deployment Strategy

The components above could be deployed and used in conjunction with a non-open-source interim system. For example, the results reporter could be used to report the election results of the interim system. The vote totaler could be used to compute the results of an election run with the interim system.

There are a number of possibilities for an open-source central ballot scanner to be used in conjunction with a non-open-source interim system. For example—

1. Open-source central ballots scanners could be responsible for scanning *all* vote-by-mail ballots, while the interim system could be responsible for counting in-precinct ballots. In addition, the interim system could even be used as a fail-safe backup in the first use in case of an unexpected issue in the open-source system.
2. Before (1), open-source scanners could be responsible for scanning *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) [add link].
3. Finally, before (2), open-source 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 test the open-source scanners. This final option can likely even be done prior to certifying the scanners.

5.2.1.2. 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.
- The components support incremental deployment. Each component can be deployed and used in conjunction with a non-open-source interim system relatively easily:

replacing individual components of an interim system and working in conjunction with other components of the interim system via import and export processes.

- 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

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 Image 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 images 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.

5.2.1.3. 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.

5.2.1.3.1. 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.

5.2.1.3.2. 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.

5.2.1.3.3. Ballot Image Interpreter (Software)

Complexity: Medium

Description. This is a software-only component responsible for interpreting ballot images, namely by generating a cast vote record (CVR) given a digital image 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 images, 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.).

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 image 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.

5.2.1.3.4. 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 images 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 Image Interpreter.
- Also needs to store digital images 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 image interpreter (see component description above).
- High-level software to orchestrate calls between the device drivers and the ballot image 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.