OPEN SOURCE VOTING SYSTEM
TECHNICAL ADVISORY COMMITTEE

ELECTIONS COMMISSION
City and County of San Francisco

Christopher Jerdonek, Chair
Larry Bafundo, Vice Chair
Carl Hage
Roan Kattouw
Tony Wasserman

Don Chan, Secretary

January 5, 2018

To: Elections Commission

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

RE: OSVTAC Report #2 (January 2018)

This is the second report of the Open Source Voting System Technical Advisory Committee (OSVTAC, or TAC) to the Elections Commission.

Below is a description of the TAC's activities since its last update on September 5, 2017. Prior updates to the Commission are available at: https://osvtac.github.io/about.

Meetings

Since its last report the TAC has had four meetings, all of which were held at 6:00 p.m. in City Hall:

- Thursday, September 21;
- Thursday, October 19;
- Thursday, November 16; and
- Thursday, December 14.

The minutes for the above meetings can be found at: https://osvtac.github.io/past-meetings.

Developments in Election Technology Modernization

The TAC has been monitoring developments in the election technology landscape, particularly when open source is being used to modernize elections systems. Colorado is conducting a risk-limiting audit using an open source solution to recount a statistical sample of ballots. Similarly,

Los Angeles County is designing a custom made voting booth and has submitted a portion of its tally system for state certification; although it is unclear if any part of its voting system will be made open source. Due to the lack of a complete combination of RFP responses, Travis County, Texas abandoned its pursuit of an open source voting system.

Building Awareness for the SF Open Source Voting Effort

At the October meeting, the TAC unanimously adopted a policy that TAC members are encouraged to make others, including the wider open source community, aware of the San Francisco open source voting project, via conferences and other venues.

Various members of the TAC attended or spoke at conferences in their personal capacities to help educate the public around the value of open source and build awareness around the modernization effort in San Francisco. Member Kattouw attended the 2nd Annual "Take Back the Vote!" National Election Integrity Conference in Berkeley, CA on October 7-8 that Chair Jerdonek spoke at. Member Bafundo spoke with Slalom, a consulting firm that is working with the City to assess the feasibility of an open source system, about how governments have used modular procurement techniques to deliver solutions with less risk.

In his personal capacity, Member Hage is participating in the National Institute of Standards and Technology (NIST) Voting Interoperability Public Working Group, which is developing election data standards for the next version of the Election Assistance Commission's (EAC's) Voluntary Voting System Guidelines (VVSG).

At the October meeting, the TAC voted to authorize two members to speak at conferences about the open source voting project as TAC members: Member Kattouw at LibrePlanet 2018 in March 2018 in Cambridge, MA and Member Wasserman at OSCON 2018 in July 2018 in Portland, OR.

Evolving Recommendations Document

The TAC continues to work on a recommendations document that is aimed at providing guidance to the City for developing an open source voting system. The document is being developed iteratively and in public view on GitHub in the same way that open source projects are developed. The current version of the document is included at the end of this report as an attachment, and future updates can be found on TAC's website here: https://osvtac.github.io/recommendations.

At the November meeting, the TAC voted to license the document under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0), as well as to license code used to generate the document under the GNU General Public License version 3 or later (GPL-3.0+). This matches the TAC's recommendations for the open source voting project itself.

Presentation by 18F on Modular Procurement

Ms. Jessie Posilkin, an innovation specialist at 18F (a consultancy within the federal government that helps agencies modernize their digital services) was invited by Member Bafundo to present to the TAC on the value of modular procurement. Ms. Posilkin leads 18F's state and local practice and shared some of her experiences helping governments adopt more agile ways of buying and developing technology. She describes an approach where governments define what they hope to achieve in terms of broad "capabilities," rather than as specific features or functionality, and engage with vendors to deliver solutions iteratively, through small and independent "modules," rather than a single contract.

Modular procurement minimizes risk to software projects in several ways. It creates flexibility for the "highest value solution" to emerge based on what is learned through development and the timeline and budget available, rather than specifying everything in a request for proposal (RFP). The latter approach often leads to unexpected costs and delays as more is learned about the problem space and the feasibility of specific aspects of the solution. A modular approach also provides the buyer with greater control over project costs and direction, as the solution is delivered iteratively, allowing for teams to demonstrate progress with stakeholders and course correct if necessary.

Applying these principles to San Francisco's open source voting effort is encouraged and the TAC plans to incorporate them in its recommendation document to the Commission. It will require a high-level vision for the new voting system, in terms of key capabilities and components, and a perspective on how the various modules to be delivered should be sequenced. This may also represent an organizational and cultural shift for the City's technology and contracting teams, as it will require greater "product ownership" as the modules are delivered iteratively from various vendors and contracts.

Attachments

1. Approved "Open Source Voting System Project Recommendations" document (dated January 2, 2018)

Open Source Voting System Project Recommendations

(Approved by OSVTAC on December 14, 2017.)

Last posted: January 2, 2018

- Introduction & Table of Contents (for multi-page version)

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License. For copyright and attribution information for this work, see this section. The source files for the text can be found on GitHub here.

Contents

1. Copyright and Attribution

This section contains the copyright and attribution information for the Open Source Voting System Technical Advisory Committee's (OSVTAC) Open Source Voting System Project Recommendations.

1.1. Copyright

Copyright (C) 2017 Larry Bafundo

Copyright (C) 2017 Carl Hage

Copyright (C) 2017 Christopher Jerdonek

Copyright (C) 2017 Roan Kattouw

Copyright (C) 2017 Tony Wasserman

1.2. Contributors

- Larry Bafundo
- Carl Hage
- Christopher Jerdonek
- Roan Kattouw
- Tony Wasserman

2. Goals

This section discusses the goals, scope, and priorities of this document and the Committee.

The TAC's Bylaws say that the TAC's purpose is to "provide technical guidance, ideas, and support to the Elections Commission on ways to improve and help ensure the success of the City and County of San Francisco's open source voting system project." The focus of TAC's effort will be on establishing parameters and recommendations to guide the future development of the voting system.

The TAC will draw on its technical expertise, the expertise of other members in the community, and from similar efforts (including other open source voting efforts) to provide guidance in areas including but not limited to open source, requirements-gathering, design, architecture, development, documentation, security, testing, certification, manufacturing, deployment, system maintenance, strategies for procurement, and project management.

2.1. Scope

1. This document will limit itself to current laws that San Francisco must satisfy, or to changes in law that San Francisco anticipates (e.g. possibly transitioning to the "vote center" model allowed by SB 450 of 2015-2016). In particular, the document will restrict itself to considering paper-ballot systems.

2. For the purposes of this document, "voting system" includes anything that is currently the responsibility of the voting system in use today. Responsibilities of a voting system include allowing voters to mark ballots (if not using pen and paper), counting ballots, reporting election results, and ensuring the integrity of the process. In addition, it may include ballot design and layout, as well as the functionality of a "remote accessible vote by mail system" as described in AB 2252 (2015-2016). It should also facilitate auditing the results of an election. The responsibilities of a voting system do not include the responsibilities of a voter registration system. The voting system may need to interoperate with the Department's EIMS® application. If the ballots are pre-printed, the voting system need not be capable of printing ballots.

2.2. Priorities

1. This document should prioritize high-level recommendations over low-level recommendations.

2. This document should prioritize recommendations that are needed sooner rather than later.

2.3. Non-goals

1. The Committee will not be designing or developing a voting system.

2. The Committee will not be drafting detailed, low-level specs that the voting system should satisfy.

3. The Committee will not be drafting an exhaustive list of requirements.

4. The Committee will not be recommending particular vendors. However, the Committee may evaluate particular *systems*.

5. The Committee will not make explicit attempts to accommodate internet voting in any form, nor voting methods not used in San Francisco. This does not preclude the Committee from recommending software designs or practices that could make such things easier to

accommodate as a side effect.

6. The Committee's recommendations will prioritize the voting system needs of San Francisco without emphasizing the needs of other jurisdictions. The needs of other jurisdictions will be considered insofar as it could help to develop and certify a system for use in San Francisco sooner (for example, if San Francisco were to collaborate with another jurisdiction and share costs). However, as stated in the previous point, this does not preclude recommending designs and practices that could make it easier to accommodate other jurisdictions.

3. Background

3.1. History of Open Source Voting in San Francisco

To provide context to the recommendations in this document, this section describes some of the history of the open source voting topic in San Francisco government.

In May 2007, the San Francisco Elections Commission passed a resolution that, among other things, established a policy that the Department of Elections give priority to voting systems that "provide the maximum level of security and transparency possible consistent with the principles of public disclosure." However, like today, no certified open source voting systems were available at that time.

In December 2007 the City, through the Department of Elections, entered into contract with Sequoia Voting Systems, Inc. for a new, proprietary voting system. In June 2010, Dominion Voting Systems, Inc. acquired Sequoia and assumed Sequoia's contract. The Department and City extended the contract with Dominion more than once. The current contract is scheduled to end at the end of 2018. The total cost of the extended contract over the eleven years (including hardware and hardware maintenance, software license fees, and election services) was approximately $22 million, with $9.6 million of that up front. This averages to around $2 million per year (see this table for an annual breakdown).

In November 2008, the Board of Supervisors passed an ordinance creating a Voting Systems Task Force (VSTF) to provide the City with recommendations on voting systems and related matters, including "models for [the] development of a voting system including proprietary, disclosed and open source software and hardware approaches."

In June 2011, the VSTF issued its final report, "Recommendations on Voting Systems for the City and County of San Francisco" (57 pages). Here are two excerpts from the recommendation text

that mention open source (from page 52):

> **2.5.4.3 Transparency, Source Code Disclosure, Licensing, and Contingency Planning**
>
> 6. The DOE should give strong preference to a voting system licensing structure that gives San Francisco all of the rights provided by an OSI-approved license, even if the system is maintained by an external party.
>
> …
>
> 8. San Francisco should be an active participant in the movement toward more open and transparent voting systems. We acknowledge the complexity of moving from the existing marketplace toward more innovative voting systems and urge San Francisco to move steadily toward the goal of transparency—even if it must do so in incremental steps.

In December 2014, the San Francisco Board of Supervisors unanimously passed a resolution supporting the creation of open source voting systems and requesting that the San Francisco Local Agency Formation Commission (LAFCo) conduct a feasibility study. In October 2015, LAFCo issued its final report, "Study on Open Source Voting Systems" (37 pages).

In November 2015, the Elections Commission unanimously passed an Open Source Voting Systems Resolution requesting that the Mayor and Board of Supervisors initiate and fund a project to develop and certify an open source voting system.

In August 2016, San Francisco Mayor Ed Lee signed the City and County of San Francisco's two-year budget for the 2016-2017 and 2017-2018 fiscal years. The budget allocated $300,000 towards the planning phase of an open source voting system project. Below are two excerpts from the proposed budget document that reference the open source voting project.

The section for the Department of Elections references the project on pages 204-205:

> As the City's current voting system nears end-of-life, the proposed budget includes $300,000 towards planning and development of a new voting system based on open source software. If completed, San Francisco would be the first City to do this. Development of an open source voting system would enable the City to own the voting system's software and to have a choice of publicly releasing it under open source license. Additionally, other jurisdictions as well as the general people could use, participate, and improve the software.

The section for the Committee on Information Technology (COIT) includes the project as one of five highlighted projects out of twenty-four, alongside initiatives like the City's new Digital

Services Team, cybersecurity, and improving the City's network (pages 447-448):

> ANNUAL PROJECTS
>
> …
>
> Over the two-year period, the proposed budget recommends $15.7 million of General Fund COIT allocation to support 24 projects. Below are a few highlighted projects.
>
> …
>
> OPEN SOURCE VOTING SYSTEM
>
> As the City's current voting system is aging, the Department of Elections is exploring an opportunity to develop a new voting system based on open source software. If completed, San Francisco would be the first city to do this. Development of an open source voting system would enable the City to own the voting system's software and have a choice of publicly releasing it under an open source license. Additionally, other jurisdictions as well as the general public could use and improve the software. The proposed budget supports initial project planning and scoping of this project.

In April 2017, the Board of Supervisors approved the City's fourth Five-Year Information & Communication Technology (ICT) Plan for Fiscal Years 2018-22. The plan included the open source voting system project among four major IT projects under consideration for the future, alongside projects like Universal Broadband and Voice over Internet Protocol (VoIP). For example, on page 11:

> However, several future projects are currently being scoped out as potentially the City's next Major IT Project, including:
>
> …
>
> **Voting System Replacement:** The Department of Elections is currently investigating alternative voting systems, including the possibility of building an open source system.

And on page 53:

> **Future Major IT Projects**
>
> In addition, the City has begun investigating what may become the next major technology project. Before beginning any new technology venture, the City recommends extensive

> planning and scoping to better understand the true cost of any new technology. The City has begun evaluating various different projects that may be considered as major investments, which include:
>
> …
>
> **Voting System Replacement:** The City's current voting system license is set to expire in 2018. Without a long-term contract in place, the City has an opportunity to pursue alternative voting systems that could promote transparency and more security. The City is currently investigating alternative options, including the possibility of building an open source system.

In April 2017, the Elections Commission voted to create an Open Source Voting System Technical Advisory Committee to "provide technical guidance, ideas, and support to the Elections Commission ('Commission') on ways to improve and help ensure the success of the City and County of San Francisco's open source voting system project." The Commission voted on the Committee's initial membership at its May meeting. The Committee was fully constituted on June 2, 2017, when the appointment of the fifth member was made final.

In May 2017, the Department of Elections issued an RFP (REG RFP #2017-01) for a contractor to "prepare a business case for developing an accessible, open source voting system." The RFP would use a portion of the $300,000 budgeted in August 2016. In September 2017, Slalom was selected as the winning bidder. Slalom's deliverable will be due in January 2018, and it will inform the City's next budget process, which will begin around that time. See here for Slalom's RFP response, the City's contract with Slalom, and Appendix A and Appendix B of the contract.

The Department of Elections' contract for its current voting system expires at the end of December 2018. The Director of Elections is aiming to lease an interim system from that point forward that can be used while an open source voting system is developed and certified. The RFP for the interim system may be issued as early as the fall of 2017.

3.2. Voting System

3.2.1. Definition

The Help America Vote Act (HAVA) of 2002 defines a voting system as follows (from 52 USC §21081: Voting systems standards):

```
(b) Voting system defined
```

```
In this section, the term "voting system" means–
(1) the total combination of mechanical, electromechanical,
 or electronic equipment (including the software, firmware,
 and documentation required to program, control, and support
 the equipment) that is used–
 (A) to define ballots;
 (B) to cast and count votes;
 (C) to report or display election results; and
 (D) to maintain and produce any audit trail information; and
(2) the practices and associated documentation used–
 (A) to identify system components and versions of such
  components;
 (B) to test the system during its development and maintenance;
 (C) to maintain records of system errors and defects;
 (D) to determine specific system changes to be made to a system
  after the initial qualification of the system; and
 (E) to make available any materials to the voter (such as
  notices, instructions, forms, or paper ballots).
```

3.2.2. Components

This section provides one possible way of listing the components of a “generic” optical-scan paper-ballot voting system. This list is not rigorous or exhaustive. Rather, it is meant for discussion purposes and to provide a sense of what functionalities are needed and how they are divided up, etc.

For simplicity, we assume that the voting system uses pre-printed ballots, as opposed to being a ballot on-demand system. We also assume that in-precinct voters are allowed to mark their ballot with a pen, as opposed to being required to interact with an electronic device. Finally, we assume the voting system includes a precinct tally, which means the system tallies the in-precinct ballots at the precinct.

The assumptions above are only for the purposes of the example illustration in this section. They were made partly because they reflect how San Francisco’s voting system works today. However, they should not be construed in any way as recommendations of the Committee or to constrain the type of voting system that San Francisco should develop. See the “Key Decisions” section below for how this list of components could change depending on certain choices.

The components in this particular list are not necessarily independent. They may overlap or contain one another. For example, the precinct ballot scanner hardware component contains a scanner device driver, the ballot picture interpreter, and the high-level scanner software

components.

Finally, note that there are many possible ways to divide a given voting system into components. For example, the granularity at which one views the system affects the number of components. We chose a mid-level granularity for this list. This lets us show how some software components are used in more than one hardware component. Differences can also result from where the "boundaries" are drawn between components (e.g. what functionalities one assigns to different components).

3.2.2.1. Hardware Components

Each of the hardware components below also needs software to function. In most cases, we list this software in the "Software Components" section.

1. Accessible Ballot-Marking Device

A device used in polling places that lets people with disabilities vote independently. It supports different accessible interfaces like audio, sip-and-puff, etc. If the computer is COTS, it may also need a custom casing or shell to increase durability and assist with polling-place transport and setup.

2. Central Ballot Scanner

A device responsible for high-speed, high-volume ballot scanning (e.g. for vote-by-mail ballots). The scanning with these machines is done in a controlled environment under staff supervision.

3. Precinct Ballot Scanner

A device used in polling places to scan and tabulate ballots cast in person. It has features like returning the ballot to the voter for possible correction if the ballot contains an overvote. Similar to the accessible device, this device may also need a custom casing or shell for durability and to facilitate polling-place use.

4. Standard laptop or desktop computers

Standard computers will also be needed for administrative tasks like ballot layout, adjudicating digital pictures of ballots, aggregating and totaling votes, and generating results reports.

3.2.2.2. Software Components

1. Voting System Database / Management

Central store (e.g. file system and/or database) and software application providing access to the voting-system information needed to conduct an election. This can include things like contest and ballot definitions, digital ballot pictures, cast vote records, and election results.

A management interface can let staff perform tasks like importing and exporting data in open data formats, adjudicating ballots that require manual inspection (e.g. ballots with write-in candidates or borderline marks)—but from the digital ballot picture rather than from the physical paper—and performing other functions needed during the canvass. This software could perhaps also provide an interface to running other software components and functions like the EIMS integration, tabulation, and results reporting.

2. EIMS® Integration.

This component is responsible for interfacing with the Department's EIMS® software. It pulls or takes election definition information exported from EIMS and imports it into the voting system database. This information can include things like what offices and candidates are on the ballot, and in what precincts, districts, and ballot styles, etc.

3. Ballot Layout

This is a software application that lets staff generate paper-ballot layouts from the election definition for each ballot type in automated or semi-automated fashion, including support for multiple languages.

4. Accessible Ballot-Marking Device Software

This is the software corresponding to the Accessible Ballot-Marking Device hardware component.

5. Ballot Picture Interpreter

This is a software library responsible for interpreting digital ballot pictures. It generates a cast vote record (CVR) from a digital picture of a ballot. This software component could potentially be used in all of the precinct scanners, the central scanners, and a software-only ballot adjudication application.

6. Scanner Device Drivers (one for precinct and one for central)

This is low-level software needed on both precinct and central ballot scanners that provides a software API to the basic hardware functionality of a ballot scanner (e.g. out-stacking a ballot, returning a ballot, advancing a ballot, etc.). This might come with COTS hardware. Separate

versions are likely needed for the precinct and central scanners.

7. Central Ballot Scanner Software

This is high-level software controlling the central ballot scanner. It interacts with the scanner device driver and ballot picture interpreter components and is responsible for things like scanning and storing digital ballot pictures, detecting the ballot layout, interpreting and tabulating ballot markings, controlling the scanner in response to the markings on a ballot, and exporting ballot data after scanning is complete.

8. Precinct Ballot Scanner Software

This component is similar to the central ballot scanner software component above and can likely share much software with it. However, it's different because it is for the case of an individual voter rather than for high-volume scanning. For example, unlike the central ballot scanner, this software will need to support returning a ballot back to the voter in the case of errors like an overvote. For the central scanner, such ballots might simply be outstacked.

8. Vote Totaler

Aggregates and counts all vote totals and generates the results in an open data format. Includes the RCV tabulation algorithm.

9. Results Reporter

Generates human-readable results reports from the results data from the vote totaler (e.g. printable results and results posted on the Department website).

3.3. Other Voting System Projects

This section includes information about some of the other voting system projects that are either (1) open source and have been or plan to be used in a US jurisdiction, or (2) are or were being developed by a jurisdiction in the US.

Special attention is paid in this section towards whether the various projects are open source because that determines whether and to what extent the source code will be available for use in San Francisco's project.

3.3.1. New Hampshire – Prime III

[TODO]

3.3.2. Los Angeles County – VSAP

Los Angeles County has been planning or working on its Voting Systems Assessment Project (VSAP) at least since 2009, when it held an event at Caltech on September 16, 2009. VSAP is a project for Los Angeles County to develop its own voting system using a "voter-centered approach." The project is led by Los Angeles County Registrar-Recorder/County Clerk (RR/CC) Dean Logan.

There is conflicting evidence as to whether any of the VSAP system will be open source and, if so, how much. On the one hand, press coverage of the project frequently mentions that the system will be open source, and Mr. Logan says it will be open source when he speaks publicly and is quoted in the media. For example, in this tweet he says, "Encouraging to see movement in this direction. #LACounty advances #opensource in #votingmodernization effort too."

Los Angeles County's April 24, 2017 VSAP RFI #17-001 also supports the view that it will be open source. For example, on page 24, it says:

> Accordingly, RR/CC is considering a Copyleft type of license such as GNU General Public License (GPL) or OSET Public License (OPL), that promotes "forever free" provisions, however it has not ruled out the use of more "permissive" open source licenses, such as the Mozilla Public License Version 2.0 (MPL), the Apache License, Version 2.0 (ALv2), the BSD 3.0 or MIT licenses. Whatever the chosen license, the transparency and ability to share the IP and the technology would need to be ensured. … LA County is seeking candid feedback from the vendor community on the intellectual property approach for VSAP.

On the other hand, there is no obvious mention of open source on VSAP's main website (e.g. on its "Principles" page). Moreover, Los Angeles County's 54-page RFP Phase 1: #17-008, which was issued five months after the RFI on September 18, 2017 to prequalify vendors, does not mention open source. The Phase 1 RFP also describes a new "Tally System" the County is working on:

> A new Tally System is required to capture and process ballot images so that vote selections on paper ballots can be digitally counted. This includes votes cast on BMD ballots at Vote Centers, as well as on Vote By Mail ballots. Similar to the ECBMS, RR/CC is currently developing the software required for the new Tally System in anticipation of a pilot in June 2018.

Los Angeles County submitted its VSAP Tally Version 1.0 to the California Secretary of State for certification on September 19, 2017. See their application for approval here (PDF, 21 pages).

However, even though the County is developing the Tally System and submitted it for certification, as of October 2017, none of the code for the Tally System appears to be publicly available, let alone open source. In addition, on page 41 of the RFP in Section 6.2 "Non-Disclosure Agreement," the RFP says—

> Prime Contractor-Led Teams who are prequalified as a result of this RFP Phase 1 will be required to sign a Non-Disclosure Agreement (NDA) as part of RFP Phase 2 prior to receiving County IP.

The requirement to sign an NDA seems inconsistent with the technology being open source.

Finally, in response to an October 5 question on Twitter about whether VSAP will be open source, Mr. Logan replied:

> Open source platform for UI and tally; publicly owned design specs and code. More detail in RFI docs at http://vsap.lavote.net

And in a second reply:

> Tally stack is all open source; details of licensing for custom code will be in Phase II RFP & was discussed in RFI; all publicly owned.

So if "platform" and "stack" refer to things like the operating system, database, programming language, etc. but not the code itself, it seems possible that none of the code will be open source but instead simply be "publicly owned." It would be helpful if Los Angeles County can provide a clearer guarantee if this interpretation isn't correct.

3.3.3. Travis County, Texas – STAR-Vote™

In 2012, Travis County, Texas started researching and designing a new voting system it called STAR-Vote™. The County spent over $330,000 in its research and design phase.

In October 2016, Travis County issued a detailed 208-page RFP covering the first phase of STAR-Vote, which was the "in-person voting module of the STAR-Vote system." The RFP made frequent reference to open source software. For example, on page 5:

> The STAR-Vote system requirements were developed from the ground up with the purpose, among other objectives, of specifying an entire voting system developed under the open source code software model.

However, the commitment to open source seemed uncertain because the RFP said the code

would start out not as open source but as disclosed source, and possibly be made open source later. For example, on page 37 (note the phrase, "with a view toward ultimately …"):

> Source code for all modules would be published, but usage rights for actual elections as well as derivative rights (as in using the code to create a derivative voting system) would be controlled by Travis County (and/or consortium) with a view toward ultimately releasing usage and derivative rights under a "suitable" (as determined by Travis County and/or consortium) open source license that would allow and encourage preparation of third-party derivative work, recognizing that voting systems must be state and federally certified;

The RFP was accompanied by an additional 16-page "Statement of Intent" document, which sought $25 million (initially a minimum of $15 million) for an entity (likely a non-profit) called the "STAR-Vote Entity."

On September 28, 2017, Travis County announced via a press release that the County would not be pursuing STAR-Vote. From their Final Report (6 pages)–

> In a nutshell, we have run into too many obstacles. There has not been enough funding, time, or support to bring STAR-Vote into the phase of being a start-up, through development and the legally-required certification process and then into use.

3.4. Resources

This section contains links to other resources and documents that may be useful for the project.

3.4.1. San Francisco

- The San Francisco Department of Elections' planning phase RFP (REG RFP #2017-01, "Preparing a Business Case for Developing an Accessible, Open Source Voting System"). In particular, see the list of links in Section I.A. starting on page 5.
- San Francisco Digital Services Team

3.4.2. Procurement

- U.S. Digital Services' TechFAR Handbook
- 18F's Modular Contracting page

3.4.3. Related Software Projects for US Government Elections

- ColoradoRLA, (Risk Limiting Audit) Project. Colorado Secretary of State. Software to upload

electronic CVRs (cast-vote-records), randomly select ballots to audit, then hand check hand selected paper ballots against stored CVRs or re-scanned paper ballots.

Contractor for open source software is Free & Fair git: ColoradoRLA, OpenRLA.

OpenCount now from Free & Fair [git] is software to tabulate scanned ballots, used with RLA when original systems do not store CVRs. [Presentation].

- Voting Systems Assessment Project (VSAP), Los Angeles County Voting station design with tablet and printer-scanner. Blank ballot sheets are inserted into printer-scanner, tablet used to make selections, printer emits printed and marked ballot for review, scanner records and feeds into collection box. Smartphone app allows pre-recorded votes to be entered via QR code. Soliciting vendors for implementation.

- Prime III, Dr. Juan E. Gilbert (now hosted at University of Florida) Tablet with docking station with keyboard and laser printer, open software. Used by NH in 2016 for accessible voting (ballot marking device). Allows home computer or phone to prepare QR code. [git]

- STAR-Vote, Travis County, TX PDF paper and slides for presentation on Travis County TX proposed system. Uses off the shelf tablet to produce printed ballot with only choices made. Scanner only reads IDs of ballots placed in box to record which ballots printed are cast. Electronic records separate. (No mail ballots.) Voters can check receipt with QR code. Demo/prototype implementation by Free & Fair.

3.4.4. Open Source Voting Organizations

- OSET Foundation 501c umbrella nonprofit to support Trust the Vote, site with actual software. [Currently, mostly Ruby-On-Rails in ruby using IEEE 1622 data models.]

  Useful diagrams of voting software architecture: (PDF, broken interactive HTML), Simpler diagram of modules.

- Open Voting Consortium Inactive (since 2011) prior effort to develop open source software. Efforts moved to CAVO.

- California Association of Voting Officials (CAVO) Nonprofit organization to promote open source voting. Election officials from several California counties are members, as well as other groups.

- Verified Voting Foundation, nonprofit to provide resources on election systems and equipment. Has links and information on voting equipment and usage across the US.

3.4.5. Election Data Standards & Organizations

- Voluntary Voting System Guidelines (VVSG). In January 2016, the U.S. Election Assistance Commission (EAC) adopted a plan where, starting in July 2017, all new voting systems would be required to be tested against the Voluntary Voting System Guidelines Version 1.1 (VVSG 1.1). The EAC approved the VVSG 1.1 in March 2015.

  The EAC was established by the Help America Vote Act of 2002 (HAVA) to develop guidance on HAVA requirements. The EAC works with NIST to sponsor Technical Guidelines Development Committee (TGDC) working groups for newer versions of the VVSG.

  [Item edited: Dec. 14, 2017 meeting.]

- Election Markup Language (EML), Original XML-based election data interchange format. Wikipedia Overview, Specifications. [2011] (Obsolete)

- IEEE VSSC/1622: Common Data Format for Election Equipment (Institute of Electrical and Electronic Engineers), Voting Systems Standards Committee). Based on EML, Superceeded by NIST SP1500.

- NIST SP1500-10x Voting Common Data Format standards. Ongoing effort on XML standards for interoperable election information. From the NIST Voting section of the Information Technology Laboratory. Coordinating and funded by EAC to produce new *Voluntary Voting Systems Guidelines*.

  Includes a good VVSG Principles and Guidelines summary.

- Voting Information Project Google/Pew effort to develop election data interchange standards, originally based on EML. Project includes collecting data from election officials nationwide. Used for Google's Civic API and third parties using Civic API. In 2016, California Secretary of State collected data from all CA counties. [2016 original contributed data is not public/open– private to Google/Pew except by special arrangement.] The VIP spec allows contest definitions, but in practice, only used for poll lookup. git

3.4.6. Additional Links

- GitHub
- Open Source Initiative (OSI)
- OpenCount

4. Facts & Assumptions

This section lists certain facts and assumptions the committee has made while drafting this document.

4.1. Facts

1. The Director of Elections' March 2017 Director's Report began outlining characteristics of the development plan for the open source voting system. These included—

   - For the system to be "Developed under version 3 of the GNU General Public License where possible, otherwise preferring similar licenses with copyleft characteristics." This is consistent with the recommendation in the Commission's Open Source Voting Systems Resolution in its third "resolved" paragraph:

     > (d) Express a preference for open source licenses with copyleft characteristics so that San Francisco and other jurisdictions can benefit from future improvements that others make to the voting system components;

   - To post the software developed for the new system "as it is written." This is also consistent with the recommendations in the same "resolved" paragraph of the Commission's resolution:

     > (b) Incorporate openness and transparency into the project, for example … by releasing all development products, including software source code and documentation, as they are developed;

4.2. Assumptions

1. The Department of Elections does not currently have the expertise to conduct the day-to-day management of the development and certification of an open source voting system.

2. The voting system should not require counting the votes on ballots by hand (not including hand-counting for audit or recount purposes).

5. Recommendations

5.1. 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.

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 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.

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 most of these components):

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

5.2.2. 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.

5.2.3. 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.

5.2.3.1. 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 prototyping and testing.

5.2.3.2. 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 prototyping and testing.

5.2.3.3. 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 prototyping and testing.

5.2.3.4. 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 prototyping and testing. Samples of ballots from past elections and/or the interim voting system.

5.2.3.5. Ballot Layout Analyzer (Hardware & Software)

Complexity: Medium

Description. This is a software component to let one "reverse engineer" structured ballot layout data from existing paper ballots from another vendor. This component may be needed during a possible interim phase in which open source components are used for scanning and interpreting ballots that are generated by a different vendor (i.e. the City's vendor during the time when the open source system is being developed). This component will be needed if that vendor is not able to provide structured ballot layout data along with the paper ballots. It is likely that this component will not be completely automated, but rather will be semi-automated.

Interfaces / data formats. Needs to accept as input:

- the "election definition" data (e.g. contests, candidates, districts, etc.).
- the digital ballot pictures.

Needs to output for each ballot type:

- the "ballot layout" data (e.g. where contests are located on each ballot card for each ballot type, etc.) that will be used as input to the Ballot Picture Interpreter component.

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

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

[Section added: Dec. 14, 2017 meeting.]

5.2.4. 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.

5.2.4.1. 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).

5.3. Requirements-gathering

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

5.3.1. 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.

5.3.1.1. 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.

5.3.1.2. 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.

5.3.1.3. 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.

5.3.1.4. 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?

5.4. Requirements

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

5.4.1. 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.

5.4.2. 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?]

5.5. Project Management

- The Department should align itself with other efforts within the City to use agile procurement and methods, and it should seek assistance where possible. Notable parts of San Francisco government beginning to use agile methods include the San Francisco Mayor's Office of Civic Innovation (MOCI) and the San Francisco Digital Services Team. See also San Francisco's *Digital Services Strategy* (PDF).

  [Item added: Dec. 14, 2017 meeting.]

- 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.]

5.6. 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.]

5.7. Procurement

[TODO]

5.8. 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.

5.9. 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.

5.10. Hardware design

[TODO]

5.11. Documentation

[TODO]

5.12. Security

[TODO]

5.13. Testing

1. **Gather real election data.** 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.

2. **Gather real digital ballot pictures.** Starting with the June 2018 election, during each election the Department should gather and save large numbers (e.g. thousands) of digital ballot pictures for future testing purposes. The Director has already expressed a willingness to do this in the case that the voting system supports it. The Department should do this during the canvass after each election because it may not be possible to obtain ballot pictures after the ballots are physically sealed and eventually destroyed. Having a variety of real-world digital ballot pictures will aid in developing and testing the ballot picture interpreter component, even if the ballot design is different from what will eventually be used. Also, using real ballots can provide test cases that might not be thought of if trying to construct test cases manually.

   [Item added: Dec. 14, 2017 meeting.]

3. **Stand-alone test data.** In the course of developing the open source voting system, where possible, structure and store test data separate from the software application (e.g. in separate repositories) and in an application-agnostic form (e.g. using open data formats). These can be

separate deliverables. The test data should include both test inputs and, when appropriate, test outputs (aka test expectations). Doing this allows the test data to be used by other applications and in particular could help facilitate additional open source implementations of components. Making the test data independent and more easily available can also improve the quality and correctness of the test data, for example by making it easier for others to check or add more test cases.

This recommendation makes more sense for higher level end-to-end tests rather than lower-level tests like unit tests since unit tests are often tied to a particular implementation. Examples of test cases for higher-level tests include things like (1) for the ballot picture interpreter component, a digital ballot picture as the input and the corresponding cast vote record as the output, and (2) for the RCV tabulator, the cast vote records for an RCV contest as the input and the round-by-round vote totals as the output.

[Item added: Dec. 14, 2017 meeting.]

5.14. Certification

[TODO]

5.15. Hardware manufacturing or assembly

[TODO]

5.16. Deployment

[TODO]

5.17. Software maintenance

[TODO]

5.18. 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.

6. FAQ

1. Is open source software more or less secure than proprietary software?

Independent studies have shown that, in general, open source software is neither more secure nor less secure than proprietary software (see for example Synopsys's "Coverity® Scan Open Source Report 2014"). Both secure and insecure open source software can be written. Similarly, both secure and insecure proprietary software can be written.

A key difference though is that, because it is publicly viewable, claims about the security of open source software can be *independently verified*, and by *anyone* (provided they have the necessary skills and time). With proprietary code, such claims can be based only on trusting those who are able to view the code.

The security of a given piece of software is primarily a function of how well the software is written. It does not (and should not) depend on keeping the code secret. The idea that software can be made secure by keeping it secret is an idea known as "security by obscurity" and is widely rejected in the security community.

Open source is already heavily used and relied upon throughout the world for security-critical applications. For example, much of the code that allows the secure transmission of information over the internet is open source.

2. How can members of the public be sure that the open source code is what is actually running on the machine?

The short answer is that there is no way to be certain that the code running on a particular device or computer is what one expects it to be (whether the software is open source or not). This is true even if very careful measures are taken. This is an extremely hard problem to solve and is an active area of research. One reason is that there is no way to be sure that the computer hardware itself can be trusted.

Having said that, good auditing practices that involve randomly checking computer results by hand against the original paper ballots are an adequate countermeasure, provided the audits are done correctly. This is why good audit procedures are important when computers are used to count ballots.

[Answer added: Dec. 14, 2017 meeting.]

3. How much of the code must be open source for the voting system to be considered open

source?

Whether something is open source or not is best answered not as a yes or no question but as a matter of degree. For example, a hardware device could be 99% open source except for one small bit of proprietary firmware.

In general, our committee recommends the approach of trying to maximize the amount of open source – i.e. the more open source, the better. There is no fundamental reason why the entire voting system can't be open source. However, if some portion isn't open source, it is better if that portion is as small as possible and if it's for optional functionality rather than required functionality.

Also, if the eventual system does contain any non-open source code, in the spirit of agile, one could work to replace that code with an open source equivalent in later versions of the system.

[Answer added: Dec. 14, 2017 meeting.]

7. Glossary

- **adjudicate**. [TODO]
- **agile**. [TODO]
- **ballot on-demand**. [TODO]
- **cast-vote record (CVR)**. [TODO]
- **central ballot scanner**. [TODO]
- **commercial off-the-shelf (COTS)**. [TODO]
- **comparison audit**. [TODO]
- **component**. [TODO]
- **digital ballot picture**. [TODO]
- **EIMS®**. EIMS® is the software application that the Department of Elections uses for certain election-related functions like maintaining voter registration data, administering polling place information, defining ballot styles, and tracking candidate filings for office (see Appendix A of the City's contract with DFM for a detailed listing of the required capabilities).

The Department signed a nine-year contract with DFM Associates for the software in June 2011 (DFM contract, Appendix A, Appendix B, Appendix C, Appendix D, Appendix E). The contract lists the per-year maintenance and support costs as ranging between $170,820.00 and $274,299.60 (see Appendix D of the contract for more detail).

- **end-to-end verifiability**. [TODO]
- **firmware**. [TODO]
- **free and open source software (FOSS)**. [TODO]
- **free software**. [TODO]
- **hardware**. [TODO]
- **hardware component**. [TODO]
- **open hardware**. [TODO]
- **open source software**. [TODO]
- **operating system**. [TODO]
- **outstack**. [TODO]
- **precinct ballot scanner**. [TODO]
- **remake**. [TODO]
- **risk-limiting audit (RLA)**. [TODO]
- **software**. [TODO]
- **software API**. [TODO]
- **software application**. [TODO]
- **software library**. [TODO]
- **software service**. [TODO]
- **software stack**. [TODO]
- **stack**. See "software stack."

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License. For copyright and attribution information for this work, see this section. The source files for the text can be found on GitHub here.

Published with GitHub Pages