Quality Assurance Plan

Relevant to: NYRT120-10

URCVT v.1.2.0 13-NY Quality Assurance Plan v.1.0.0 document is solely for use in the State of New York. This document can be expanded or updated as is necessary or required. Any recommendations listed in this document should not supersede user jurisdiction procedures or other controlling governance entities.

URCVT v.1.2.0 13-NY Quality Assurance Plan v.1.0.0

The URCVT software is developed with the following tools, policies, and practices to ensure robust software quality and reliability. URCVT testing relates only to the function of the software and performs no parts and materials testing as we produce only software and do not design or manufacture hardware.

1) Version Control Software:
We use Git version control software (git-scm.com) in conjunction with Github (github.com/BrightSpots/rcv) to coordinate the efforts of our developers and maintain a complete record of ALL software code changes to the URCVT and the reasoning behind them.

2) Software Revision Control Branching Policy:
To isolate code in development from production code (released), we use Git-Flow: a branching policy built on top of Git. The policy coordinates development, testing, review, and deployment of code throughout the development life-cycle. Details can be found here: (www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow)

3) Code reviews:
All code changes submitted for incorporation into the URCVT software must undergo a manual code review from at least one developer other than the original drafter/developer. Other stakeholders and experts are involved with code reviews as needed. Code reviews offer an additional opportunity to identify potential defects, improve code structure, clarity, performance, and robustness. Code merges are blocked until at least one developer explicitly approves the code submission, at which point the code is merged, and regression tests will be run. For code review examples, see: github.com/BrightSpots/rcv/pulls.

4) Regression Tests:
We have developed a suite of 57 Tabulation Regression Tests. We continue to add more tests as new features and bug fixes are added. These tests are designed to verify that new code changes do not inadvertently alter any tabulation results.

Each test has a set of inputs (just like any tabulation): A config file and CVR file(s). Additionally, each test has an expected results file. The automated test will: load the config, run the tabulation, verify that the tabulation output matches the expected results. For actual test data, see:
github.com/BrightSpots/rcv/tree/master/src/test/resources/network/brightspots/rcv/test_data.

The names of the tests are mostly self-explanatory. For example, *test_set_1_exhaust_at_overvote* tests whether the tabulator correctly interprets the overvoteRule = "exhaust immediately" setting. Some of the tests aren't testing specific features but instead are testing a known data set, e.g., *2018_maine_governor_primary*. The complete list is included below.

All regression tests are run by developers and must pass before any new code can be merged into the tabulator.

5) System Requirements Testing:
Minimum system requirements were determined by repeated tabulation of elections with 100,000 CVRs, 1,000,000 CVRs, and 6,000,000 CVRs on each target platform. These tests were timed to ensure they complete in a reasonable amount of time successfully.

6) Ad-hoc user testing:
When developing new features, we try to recruit as much user feedback and user testing as possible. These testers are typically from the RCVRC staff, vendor partners, election administrators, and RCV activists.

7) Process for Handling of deficiencies:
Any defects discovered through testing or reported by users are recorded and tracked using Github issue tracking tools. We confirm the existence of the defect, evaluate its severity, and mark it accordingly. Depending on user impact, development resources, and release timelines, we schedule developers to address the defects in order of priority and then do the actual work. Typically, this involves:

1) reproducing the defect
2) making necessary code changes to fix the defect on an isolated git branch
3) requesting a code review and implementing any changes arising from the code review
4) verifying that regression tests pass
5) pushing the code change to the main branch and linking the issue in the commit message
6) closing the issue and linking to the commit in Github issue tracker

In this way, we are able to ensure that all known issues are tracked, and the most important ones are mitigated according to criticality. For more details, see: https://github.com/BrightSpots/rcv/issues.

8) Quality Conformance Inspections and Documentation:
We recognize that our process for test inspections, records, and documentation is not fully vetted. As such, we have a roadmap for adding automated testing tools to our build system in the summer of 2021. This will address these requirements in a robust and reliable way.

All documentation in this submission is named according to the formula:
[System version] [Document Number-NY] [Document Name] [Document version]
System Version: URCVT 1.2.0
Document version: 1.0.0 (all documents are new with this submission)
Example:
URCVT 1.2.0 11-NY Quality Assurance Plan 1.0.0

9) URCVT Regression Test Listing:

```
@DisplayName("NIST XML CDF 2")
@DisplayName("unisyn_xml_cdf_city_tax_collector")
@DisplayName("unisyn_xml_cdf_city_mayor")
@DisplayName("unisyn_xml_cdf_city_council_member")
@DisplayName("unisyn_xml_cdf_city_chief_of_police")
@DisplayName("unisyn_xml_cdf_city_coroner")
@DisplayName("unisyn_xml_cdf_county_sheriff")
@DisplayName("unisyn_xml_cdf_county_coroner")
@DisplayName("Clear Ballot - Kansas Primary")
@DisplayName("Hart - Travis County Officers")
@DisplayName("Hart - Cedar Park School Board")
@DisplayName("Dominion test - Alaska test data")
@DisplayName("Dominion test - Kansas test data")
@DisplayName("Dominion test - Wyoming test data")
@DisplayName("Dominion - No Precinct Data")
@DisplayName("test invalid params in config file")
@DisplayName("test invalid source files")
@DisplayName("2015 Portland Mayor")
@DisplayName("2015 Portland Mayor Candidate Codes")
@DisplayName("2013 Minneapolis Mayor Scale")
@DisplayName("Continue Until Two Candidates Remain")
@DisplayName("Continue Until Two Candidates Remain with Batch Elimination")
@DisplayName("2017 Minneapolis Mayor")
@DisplayName("2013 Minneapolis Mayor")
@DisplayName("2013 Minneapolis Park")
@DisplayName("2018 Maine Governor Democratic Primary")
@DisplayName("testMinneapolisMultiSeatThreshold")
@DisplayName("test for overvotes")
@DisplayName("test excluding candidates in config file")
@DisplayName("test minimum vote threshold setting")
@DisplayName("test skipping to next candidate after overvote")
@DisplayName("test Hare quota")
@DisplayName("test sequential multi-seat logic")
@DisplayName("test bottoms-up multi-seat logic")
@DisplayName("test bottoms-up multi-seat with threshold logic")
@DisplayName("test allow only one winner per round logic")
@DisplayName("precinct example")
@DisplayName("missing precinct example")
@DisplayName("test tiebreak seed")
@DisplayName("skipped first choice")
@DisplayName("exhaust at overvote rule")
@DisplayName("overvote skips to next rank")
@DisplayName("skipped choice exhausts option")
@DisplayName("skipped choice next option")
@DisplayName("two skipped ranks exhausts option")
@DisplayName("duplicate rank exhausts")
@DisplayName("duplicate rank skips to next option")
```

```
@DisplayName("multi-cdf tabulation")
@DisplayName("multi-seat whole number threshold")
@DisplayName("multi-seat fractional number threshold")
@DisplayName("tiebreak using permutation in config")
@DisplayName("tiebreak using generated permutation")
@DisplayName("tiebreak using previousRoundCountsThenRandom")
@DisplayName("treat blank as undeclared write-in")
@DisplayName("undeclared write-in (UWI) cannot win test")
@DisplayName("multi-seat UWI test")
@DisplayName("overvote delimiter test")
```

Document Revision History

Date	Version	Description	Author
04/20/2021	1.0.0	Quality Assurance Plan	Jon Moldover