# A blue and white logo AI-generated content may be incorrect.

# Technical Specification

## SPC Station Health Charts

**Version:** 1.0

**Date:** October 6, 2025

**Status:** Implemented

## 1. Technical Overview

### 1.1 Technology Stack

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Technology** | **Version** | **Rationale** |
| **Backend Language** | Python | 3.7+ | Pre-installed enterprise-wide; no admin rights needed |
| **Web Server** | http.server | stdlib | Zero dependencies; sufficient for localhost |
| **Frontend** | HTML5 + Vanilla JS | - | Universal browser support; no build step |
| **Charting** | HTML5 Canvas API | - | No external libraries; full PNG export control |
| **Data Format** | CSV | - | Universal; easy to generate |
| **API Protocol** | HTTP/JSON | - | Standard web protocols |

### 1.2 System Architecture

┌─────────────────────────────────────────────────────┐ │ User's Desktop │ │ │ │ ┌────────────┐ ┌──────────────────┐ │ │ │ Browser │◄───────►│ Python Server │ │ │ │ (Client) │ HTTP │ (localhost:8000)│ │ │ └────────────┘ └──────────────────┘ │ │ │ │ │ │ │ │ │ │ v v │ │ ┌────────────┐ ┌──────────────────┐ │ │ │ Canvas │ │ SPC Processor │ │ │ │ Rendering │ │ (Statistics) │ │ │ └────────────┘ └──────────────────┘ │ │ │ │ │ v │ │ ┌──────────────────┐ │ │ │ CSV Files │ │ │ │ (input/) │ │ │ └──────────────────┘ │ └─────────────────────────────────────────────────────┘

## 2. Backend Implementation

### 2.1 server.py - HTTP Request Handler

**File:** server.py  
**Lines of Code:** ~200  
**Dependencies:** Python stdlib only

#### Class: SPCHandler

class SPCHandler(SimpleHTTPRequestHandler):

"""

Custom HTTP request handler for SPC Dashboard.

Routes:

- GET / → Serve dashboard\_standalone.html

- POST /api/process → Process uploaded CSV

- POST /api/load-actual → Load sample data from input/

- OPTIONS \* → CORS preflight

"""

#### Key Methods

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Purpose** | **Input** | **Output** |
| end\_headers() | Add CORS headers | None | HTTP headers |
| do\_OPTIONS() | Handle CORS preflight | HTTP request | 200 OK |
| do\_GET() | Serve HTML file | GET / | HTML content |
| do\_POST() | Process API requests | JSON body | JSON response |

### 2.2 spc\_processor.py - Statistical Engine

**File:** spc\_processor.py  
**Lines of Code:** ~400  
**Purpose:** Core SPC calculations and phase detection

#### Function: parse\_csv(csv\_text)

**Input:** Raw CSV string  
**Output:** List of dictionaries [{station, measure, date, value}, ...]

def parse\_csv(csv\_text: str) -> List[Dict[str, Any]]:

"""

Parse CSV text into structured data points.

Expected format: station,measure,date,value

Returns:

List of dicts with keys: station, measure, date, value

"""

reader = csv.DictReader(StringIO(csv\_text))

data\_points = []

for row in reader:

try:

data\_points.append({

'station': row['station'].strip(),

'measure': row['measure'].strip(),

'date': row['date'].strip(),

'value': float(row['value'])

})

except (KeyError, ValueError):

continue # Skip invalid rows

return data\_points

#### Function: calculate\_initial\_limits(values)

**Input:** List of numeric values  
**Output:** Tuple (center\_line, moving\_range\_average)

**Algorithm:**

1. Calculate center line (CL) = mean of values
2. Calculate moving ranges: mR[i] = |value[i] - value[i-1]|
3. Calculate average moving range: mR̄ = mean(mR)
4. Return (CL, mR̄)

def calculate\_initial\_limits(values: List[float]) -> Tuple[float, float]:

"""

Calculate center line and moving range average.

Wheeler's method:

- CL = mean(X)

- mR̄ = mean(|X[i] - X[i-1]|)

"""

if len(values) < 2:

return (values[0] if values else 0, 0)

cl = sum(values) / len(values)

moving\_ranges = [abs(values[i] - values[i-1])

for i in range(1, len(values))]

mR\_bar = sum(moving\_ranges) / len(moving\_ranges)

return (cl, mR\_bar)

**Control Limit Calculation:**

# Natural Process Limits (2.66 sigma)

ucl = cl + 2.66 \* mR\_bar

lcl = max(0, cl - 2.66 \* mR\_bar) # Non-negative

#### Function: find\_phase\_end(values, cl, ucl, lcl)

**Input:** List of values, control limits  
**Output:** Index where phase ends (signal detected)

**Wheeler's Rules for Detecting Special Causes:**

* **Rule #1:** Point beyond control limits (outside UCL/LCL)
* **Rule #4:** 8 consecutive points on one side of centerline

def find\_phase\_end(values: List[float], cl: float,

ucl: float, lcl: float) -> int:

"""

Find where phase ends using Wheeler's Rules.

Returns:

Index where current phase should end (signal detected)

"""

RUN\_LENGTH = 8

consecutive\_above = 0

consecutive\_below = 0

for i, value in enumerate(values):

# RULE #1: Point outside control limits

if value > ucl or value < lcl:

# Signal! Phase ends at the point BEFORE this outlier

return max(0, i - 1)

# RULE #4: Track 8-point runs

if value > cl:

consecutive\_above += 1

consecutive\_below = 0

elif value < cl:

consecutive\_below += 1

consecutive\_above = 0

# Signal detected: 8 consecutive on one side

if consecutive\_above >= RUN\_LENGTH or consecutive\_below >= RUN\_LENGTH:

# Phase ends at the point BEFORE the run started

return max(0, i - RUN\_LENGTH)

# No signal found - entire dataset is one phase

return len(values) - 1

#### Function: detect\_phases(data\_points)

**Input:** List of data points (sorted by date)  
**Output:** Dictionary with augmented points and phase metadata

**Proper SPC Phase Detection Algorithm:**

1. Start with baseline data (minimum 20 points)
2. Calculate limits from ONLY that baseline
3. Extend phase forward, watching for signals
4. When signal detected, start new phase at signal point
5. Recalculate limits from actual phase data
6. Augment each point with its phase's control limits

### 2.3 load\_actual\_data.py - Multi-File CSV Loader

**File:** load\_actual\_data.py  
**Lines of Code:** ~150  
**Purpose:** Load and convert multiple CSV files from input/ folder

#### Station Name Mapping

STATION\_MAP = {

'Austin': 'AUS',

'Dallas': 'DAL',

'Dallas Love Field': 'DAL',

'Houston': 'HOU',

'Houston Hobby': 'HOU'

}

MEASURE\_NAMES = {

'maintenance\_cancels.csv': 'Maintenance Cancels',

'maintenance\_delays.csv': 'Maintenance Delays',

'scheduled\_maintenance\_findings.csv': 'Scheduled Maintenance Findings',

'unscheduled\_maintenance.csv': 'Unscheduled Maintenance'

}

#### Function: convert\_to\_spc\_format(input\_folder)

**Input:** Path to folder containing CSV files  
**Output:** Combined CSV string in standard format

**Conversion Process:**

1. Scan folder for .csv files
2. For each file:
   * Read CSV (format: timestamp,station,metric\_value)
   * Infer measure name from filename
   * Map station names using STATION\_MAP
   * Convert to standard format (station,measure,date,value)
3. Combine all files into single CSV string
4. Return combined CSV

## 3. Frontend Implementation

### 3.1 dashboard\_standalone.html - Single-Page Application

**File:** dashboard\_standalone.html  
**Lines of Code:** ~900  
**Structure:** Self-contained HTML with embedded CSS and JavaScript

#### HTML Structure

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>SPC Station Health Charts</title>

<style>

/\* Embedded CSS - ~150 lines \*/

</style>

</head>

<body>

<div id="header">...</div>

<div id="controls">...</div>

<div id="content">

<!-- Charts render here -->

</div>

<script>

// Embedded JavaScript - ~700 lines

</script>

</body>

</html>

### 3.2 JavaScript Architecture

#### Global State

let chartData = null; // Current dataset

let selectedStation = 'all'; // Station filter

#### Key Functions

|  |  |  |
| --- | --- | --- |
| **Function** | **Purpose** | **Lines** |
| loadActualData() | Load test data from /api/load-actual | ~20 |
| handleFileUpload(event) | Process CSV file upload | ~30 |
| updateStationSelect(stations) | Populate station dropdown | ~15 |
| renderCharts() | Render all charts for selected station | ~30 |
| drawChart(canvasId, data, title) | Draw single chart on canvas | ~200 |
| downloadChart(canvasId, chartName) | Export chart as PNG | ~10 |
| showError(message) | Display error message | ~10 |

#### Function: drawChart(canvasId, data, title)

**Core Rendering Steps:**

1. Clear canvas
2. Calculate scales (X and Y axes)
3. Draw axes
4. Draw control limits (UCL, CL, LCL)
5. Draw phase boundaries (vertical dotted lines)
6. Draw data line
7. Draw data points (blue for normal, red for out-of-control)
8. Draw title
9. Draw axis labels

#### Function: downloadChart(canvasId, chartName)

**PNG Export Implementation:**

function downloadChart(canvasId, chartName) {

const canvas = document.getElementById(canvasId);

if (!canvas) return;

// Convert canvas to PNG data URL

const dataURL = canvas.toDataURL('image/png');

// Create temporary download link

const link = document.createElement('a');

link.download = `${chartName.replace(/[^a-zA-Z0-9]/g, '\_')}.png`;

link.href = dataURL;

link.click();

}

## 4. Data Structures

### 4.1 JSON API Response

{

"success": true,

"chartData": {

"DAL": {

"Maintenance Cancels": {

"points": [

{

"station": "DAL",

"measure": "Maintenance Cancels",

"date": "2023-01-02",

"value": 3.2,

"ucl": 5.1,

"cl": 3.0,

"lcl": 0.9

}

],

"phases": [

{

"startIndex": 0,

"endIndex": 45,

"cl": 3.0,

"ucl": 5.1,

"lcl": 0.9,

"phaseNumber": 1

}

]

}

}

},

"stations": ["AUS", "DAL", "HOU"]

}

## 5. Performance Optimizations

### 5.1 Backend Optimizations

|  |  |  |
| --- | --- | --- |
| **Optimization** | **Technique** | **Benefit** |
| **List Comprehensions** | Use Python list comprehensions for loops | 2-3x faster than traditional loops |
| **Single-Pass Parsing** | Parse CSV once, group in single loop | Avoid multiple passes over data |
| **Minimal Data Transfer** | Round decimals to 2 places | Reduce JSON payload size |

### 5.2 Frontend Optimizations

|  |  |  |
| --- | --- | --- |
| **Optimization** | **Technique** | **Benefit** |
| **Canvas Rendering** | Use Canvas API instead of DOM | GPU accelerated, no reflows |
| **Debounced Resize** | Delay redraw until resize stops | Prevent excessive redraws |
| **Lazy Loading** | Only render visible charts | Faster initial load |

## 6. Security Considerations

### 6.1 Threat Model

**Risk Level:** Low (localhost-only desktop tool)

|  |  |  |
| --- | --- | --- |
| **Threat** | **Status** | **Mitigation** |
| CSV Injection | Accepted | No Excel output; Canvas rendering prevents XSS |
| Path Traversal | Mitigated | Only read file content, not paths |
| XSS | Mitigated | Canvas rendering (not innerHTML) |
| DoS (Large CSV) | Accepted | User's own machine; reasonable limits |

## 7. Testing Strategy

### 7.1 Manual Testing (Performed)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Description** | **Expected** | **Actual** | **Status** |
| T01 | Load test data | 24 charts display | 24 charts | ✅ Pass |
| T02 | Upload Format A CSV | Charts display, filename = measure | Correct | ✅ Pass |
| T03 | Upload Format B CSV | Charts display, measure from column | Correct | ✅ Pass |
| T04 | Upload invalid CSV | Error message | "No valid data points" | ✅ Pass |
| T05 | Export PNG | File downloads | PNG in Downloads folder | ✅ Pass |
| T06 | Phase detection | Phases marked correctly | Matches Wheeler's Rules | ✅ Pass |
| T07 | Station filter | Only selected station charts | Correct filtering | ✅ Pass |
| T08 | Port conflict | Clear error message | Address in use error | ✅ Pass |

## 8. Deployment

### 8.1 Build Process

**No build step required!**

* No transpilation (no Babel/TypeScript)
* No bundling (no Webpack/Vite)
* No minification
* No dependency installation

### 8.2 Distribution Package Contents

SPC-Station-Health-Charts.zip

├── dashboard\_standalone.html # Main UI (self-contained)

├── server.py # HTTP server + routing

├── spc\_processor.py # Statistical calculations

├── load\_actual\_data.py # CSV data loader

│

├── START\_DASHBOARD.bat # Windows launcher

├── start\_dashboard.sh # Mac/Linux launcher

│

├── README.md # Technical README

├── README\_DISTRIBUTION.md # User-facing README

├── .gitignore # Git ignore rules

│

├── input/ # Sample CSV data

│ ├── maintenance\_cancels.csv

│ ├── maintenance\_delays.csv

│ ├── scheduled\_maintenance\_findings.csv

│ └── unscheduled\_maintenance.csv

│

└── output/ # Exported PNG charts

## 9. Version History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Changes** | **Lines Changed** |
| 0.1 | 2025-09-28 | Initial React + Recharts version | N/A |
| 0.2 | 2025-10-01 | Fixed infinite render loop | ~20 |
| 0.3 | 2025-10-02 | Switched to standalone HTML + Canvas | ~900 (rewrite) |
| 0.4 | 2025-10-04 | Fixed phase detection (Wheeler's Rules) | ~150 |
| 0.5 | 2025-10-05 | Added XmR charts (both X and mR) | ~100 |
| **1.0** | **2025-10-06** | **Production release** | **~1600 total** |

## Approval

|  |  |  |  |
| --- | --- | --- | --- |
| **Role** | **Name** | **Date** | **Signature** |
| Tech Lead | Development Team | 2025-10-06 | ✅ Approved |
| Code Reviewer | Senior Developer | 2025-10-06 | ✅ Reviewed |

**Document Version History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Changes** |
| 1.0 | 2025-10-06 | Development Team | Initial technical specification |

*Southwest Airlines - Technical Operations Analytics Team*